MANTRA I: An SIMD Processor Array for Neural Computation

Marc A. Viredaz
Swiss Federal Institute of Technology
EPFL – LAMI, IN-F Ecublens, CH – 1015 Lausanne
E-mail: viredaz@di.epfl.ch

Abstract

This paper presents an SIMD processor array dedicated to the implementation of neural networks. The heart of this machine is a systolic array of simple processing elements (PEs). *A VLSI custom chip containing 2×2 PEs was built. The machine is designed to sustain sufficient instruction and data flows to keep a utilization rate close to 100 %. Finally, this computer is intended to be inserted in a network of heterogeneous nodes.*

1 Introduction

Artificial neural networks have experienced, in the past ten years, a very rapidly growing interest. A large part of the researches and applications in this domain however still relies on simulations. Many researchers and a few commercial companies have proposed specialized hardware implementations. The majority of these are however dedicated to a single neural-network model or algorithm. At the present day, there is no platform which is, at the same time, versatile enough to implement any model, and fast enough to be used on large problems. In this paper a dedicated SIMD processor array is proposed as a *multi-model neural computer*.

This machine is based on a 2-D systolic array of VLSI custom chips. The whole machine is designed to sustain a utilization rate of the array close to 100 %. Several features were added for this purpose. A large part of the hardware is dedicated to an efficient internal storage and to input/output. Finally, provision has been made to integrate this machine in a heterogeneous-node network.

The MANTRA I machine has been designed with the following models in mind:

- Mono-layer networks: Perceptron [1], ADALINE [2], and delta rule.
- Multi-layer feed-forward networks: back-propagation rule [3].
- Fully-connected recurrent networks: Hopfield model [4].
- Self-organizing feature maps: Kohonen model [5].

For most of these algorithms, different versions can be run. It should however be stressed that the machine does not hard-wire these algorithms, but provides a set of basic operations. Any model making use of only these operations, can therefore be implemented.

A description of these algorithms is beyond the scope of this paper. However, to appreciate the proposed architecture from the computer architect point-of-view, it is sufficient

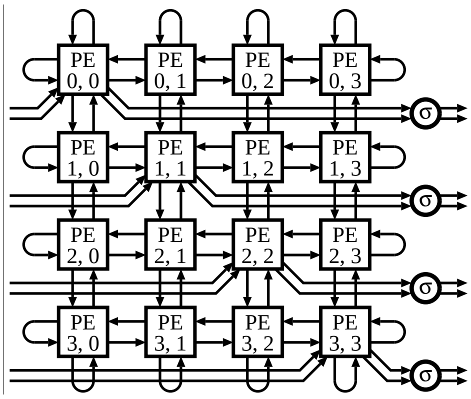

Figure 1 : Square systolic array of GENES IV processing elements.

to know what kind of computation should be performed. For this purpose, appendix A.1 describes the back-propagation rule, and appendix A.2 a version of the Kohonen model. The delta rule can be viewed as a single-layer back-propagation.

In collaboration with other laboratories, this experimental machine will be applied to static security assessment in power distribution networks [6] and to meteorology.

2 The GENES IV systolic array

In his Ph.D. thesis [7], F. Blayo observed that four basic scalar and matrix operations are sufficient to implement most widely-used neural-network models. He showed that these operations can be efficiently implemented on a 2-D systolic array. The same analysis was then extended to a larger set of models in a paper by C. Lehmann *et al.* [8], resulting in nine operations.[1] The computer architect should however observe that some operations can always be combined, hence allowing a smaller set of "composed operations."

Based on these considerations, a few VLSI digital circuits called *Generic Element for Neuro-Emulator Systolic arrays* or *GENES* were implemented. The first prototype, referred to as *GENES HN8* [9], was used to show that a such array can be effective. The latest version, called *GENES IV*, forms the heart of the MANTRA I machine. Since this chip has already been described elsewhere [10] only a short overview is provided here.

A GENES IV array is a square mesh of simple *processing elements (PEs)*. Each PE is connected by serial lines to its four neighbors as shown in figure 1. All input and output operations are performed by the PEs located on the north-west to south-east diagonal.

The GENES IV structure implements six different operations, grouped in three categories. The first operation is the *matrix-vector product*:

$$\vec{\mathbf{p}} \;=\; \mathbf{W}\cdot\vec{\mathbf{x}} \tag{1}$$

This operation can also be viewed as the *scalar* or *dot product* between the input vector $\vec{\mathbf{x}}$ and each row $\mathbf{W}_i$ of the synaptic weight matrix $\mathbf{W}$. The second operation is the computation of the *squared Euclidean distance* between a vector and each row of a matrix:

$$p_i \;=\; \sum_{j=1}^{N}(x_j - W_{i,j})^2 \tag{2}$$

The systolic flow of data for these two operations is shown in figure 2(a). Each cell

[1]It should be noticed that some of the original operations proposed in [7] were divided or decomposed into several operations in [8], in an attempt to provided a more rigorous analysis.

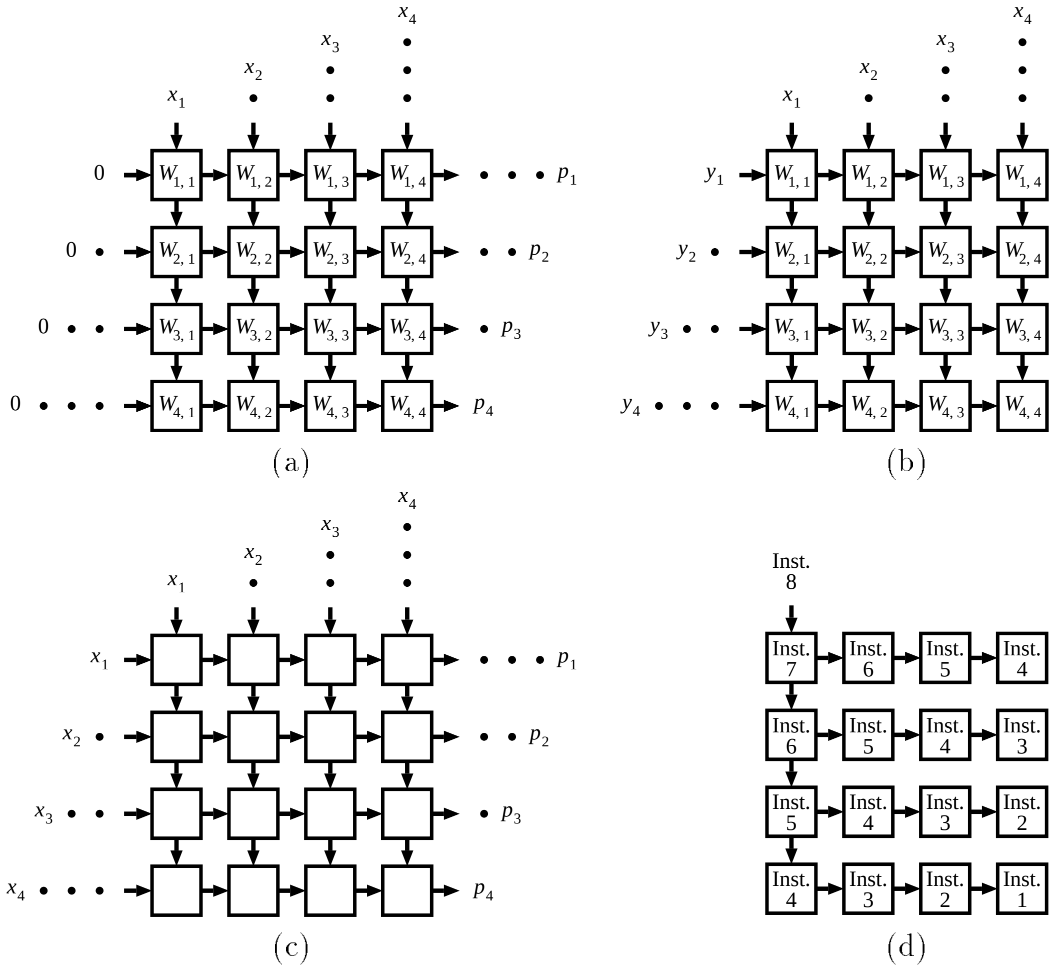

Figure 2 : Systolic flows in the GENES IV array. (a) Matrix-vector product or Euclidean distance computation. (b) Synaptic weight updating using the Hebbian or Kohonen learning rules. (c) Search for the smallest/largest element of a vector. (d) Instruction flow.

contains an element $W_{i,j}$ of the matrix $\mathbf{W}$. The input vector $\vec{\mathbf{x}}$ arrives from the north. Each PE computes either the product $x_j \cdot W_{i,j}$ of its north input and the stored weight or the squared difference $(x_j - W_{i,j})^2$ of the same two values. This result is then added to the partial sum received from the west, and sent to the east. The partial sums are initially set to zero. For the sake of clarity, figure 2 shows the input values "diagonally" entered from the north and west sides, and the output vector retrieved from the east one. In the actual operation, all inputs are entered on the diagonal, and initially flow in the reverse direction until they are reflected. In the same way, the outputs are reflected on the east side and then extracted from the diagonal. This feature allows to enter and retrieve all the elements of a vector at the same time.

The second category groups the synaptic weight updating by the *Hebbian learning rule* and the *Kohonen learning rule*, described by equation (3) and (4) respectively:

$$W_{i,j} \; := \; W_{i,j} + y_i \cdot x_j \tag{3}$$
$$W_{i,j} \; := \; W_{i,j} + y_i \cdot (x_j - W_{i,j}) \tag{4}$$

Equation (3) can be written in a matrix form as $\mathbf{W} := \mathbf{W} + \vec{\mathbf{y}} \cdot \vec{\mathbf{x}}^{\mathrm{T}}$. The data flow for both operations is shown in figure 2(b). Two vectors $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$ flow from the north and west. On each PE, the stored synaptic weight is updated according to equation (3) or (4).

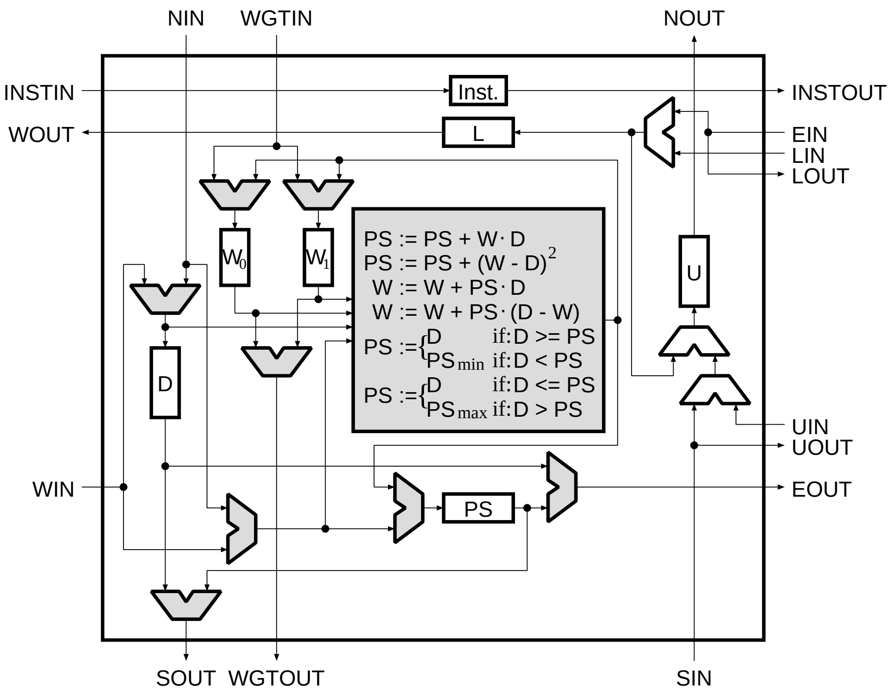

Figure 3: Architecture of a GENES IV diagonal PE. On non-diagonal PEs, the input/output signals: UIN, LIN, UOUT, and LOUT are disabled. Shaded units are controlled by the instruction word, while other units are globally controlled by external signals.

The last two operations are the search for the largest and smallest element of a vector:

$$y_i = \begin{cases} x_i & \text{if: } x_i = \max_j(x_j) \\ N_{\min} & \text{if: } x_i < \max_j(x_j) \end{cases} \tag{5}$$

$$y_i = \begin{cases} x_i & \text{if: } x_i = \min_j(x_j) \\ N_{\max} & \text{if: } x_i > \min_j(x_j) \end{cases} \tag{6}$$

Where $N_{\min}$ is the smallest representable number, and $N_{\max}$ the largest one. Figure 2 (c) shows the flow of data for these operations. The vector $\vec{\mathbf{x}}$ is injected from the north and west. Each PE forwards the data received from the west if it is larger (smaller) than that received form the north, otherwise it replaces it with $N_{\min}$ ($N_{\max}$).

Like any systolic array, a GENES IV system behaves as a pipeline. The performance, may therefore dramatically decrease if it is not kept full. When the operations have to be changed often,[2] as it is the case with neural algorithms, it would be disastrous to have every PE executing the same instruction at the same time (conventional SIMD paradigm), because the pipeline would have to be emptied and refilled for each change. To avoid this inefficiency, instructions flow as shown by figure 2 (d), accompanying the corresponding data. This feature ensures that the pipeline is kept full during any possible computation.

Figure 3 shows the resulting PE. An important feature is that two registers W_0 and W_1 can hold the synaptic weights. This characteristic gives to the GENES IV architecture the ability to deal with virtual matrixes larger than the physical array. The virtual matrix is divided into sub-matrixes that fit on the array, which is then time-shared between them. A sub-matrix can therefore be used for computation while the previous one is saved to memory and the next one simultaneously loaded, maintaining a 100 % utilization rate.

[2]In the present context, "often" means: after a number of operations that is smaller or of the same order of magnitude than twice the number of PEs per side (pipeline depth).

Model	Evaluation phase	Learning phase
Perceptron, ADALINE, Delta rule	$400 \cdot 10^6$ CPS	$200 \cdot 10^6$ CUPS
Back-propagation rule	$400 \cdot 10^6$ CPS	$133 \cdot 10^6$ CUPS
Kohonen with minimum/maximum	$200 \cdot 10^6$ CPS	$100 \cdot 10^6$ CUPS

Table 1 : Peak performance of a 40×40 PE GENES IV array (MANTRA I machine).

Another particularity is the use of multiplexers to exchange the functionality of the rows and columns. The transpose matrix $\mathbf{W}^T$ can therefore be used for the matrix-vector multiplication (or any other operation) as required by the back-propagation rule.

A VLSI chip, containing 2×2 PEs, was designed with standard cells in a CMOS $1\,\mu$m technology. It is composed of 71690 transistors (3179 standard cells) on a die of $6.3 \times 6.1\,\text{mm}^2$. The registers W_0 and W_1 are 33 bits wide (including an overflow bit), but only the most significant 16 bits are used for non-learning operations. The other registers D, PS, U, and L are all 40 bits wide. Only 16 bits of the register D are however used for other operations than the search for the largest/smallest element of a vector. Similarly, the register PS is used as a 17-bit value for the two learning operations. Finally, since the computation is serial, a PE requires 40 clock cycles to complete an instruction. This period is referred to as a *macro-cycle*.

The heart of the MANTRA I machine, shown in figure 4, is a GENES IV array of 40×40 PEs running at 10 MHz. Like any SIMD computer, this machine is composed of a parallel or SIMD part, described in section 3, and of a controller, discussed in section 4. The peak performance of a such array is given in table 1

3 The SIMD part

3.1 The input/output units

Neural networks require different types of data: (1) the synaptic weights $\mathbf{W}$, (2) the inputs $\vec{\mathbf{x}}$, (3) the outputs $\vec{\mathbf{y}}$, and (4) the desired outputs $\vec{\mathbf{d}}$. Since the outputs $\vec{\mathbf{y}}$ of an iteration is often the inputs $\vec{\mathbf{x}}$ of the next one, both should be stored in the same memory. Furthermore, during weight updating phases, two such values must be simultaneously input to the array. This imposes the use of two memory banks, referred to as *XY memory* and *auxiliary Y memory*. Other data can be stored in separate memories, namely the *weight memory* and *desired output memory*.

The weight, XY, and auxiliary Y memories must support a read and a write access every clock cycle. At the same time, the controller should be able to store or retrieve data without disturbing the SIMD part. These two requirements make multi-port memories good candidates. However, only two types of these devices can be found as conventional off-the-shelf chips: dual-port static RAMs and Video RAMs. The first type was discarded because of its small capacity and large package, and the second one because there is no way to remove one of the serial/parallel conversion units by using their serial port.[3]

FIFOs were instead used to let the controller store and retrieve data. Inputs can be read from a FIFO or from memory. In the former case they can also be simultaneously written into the corresponding memory for a later use. Similarly outputs can be written into a FIFO, into memory, or into both at the same time.

[3]This is due to the parallel nature of the sigma and function of Y units described in section 3.2.

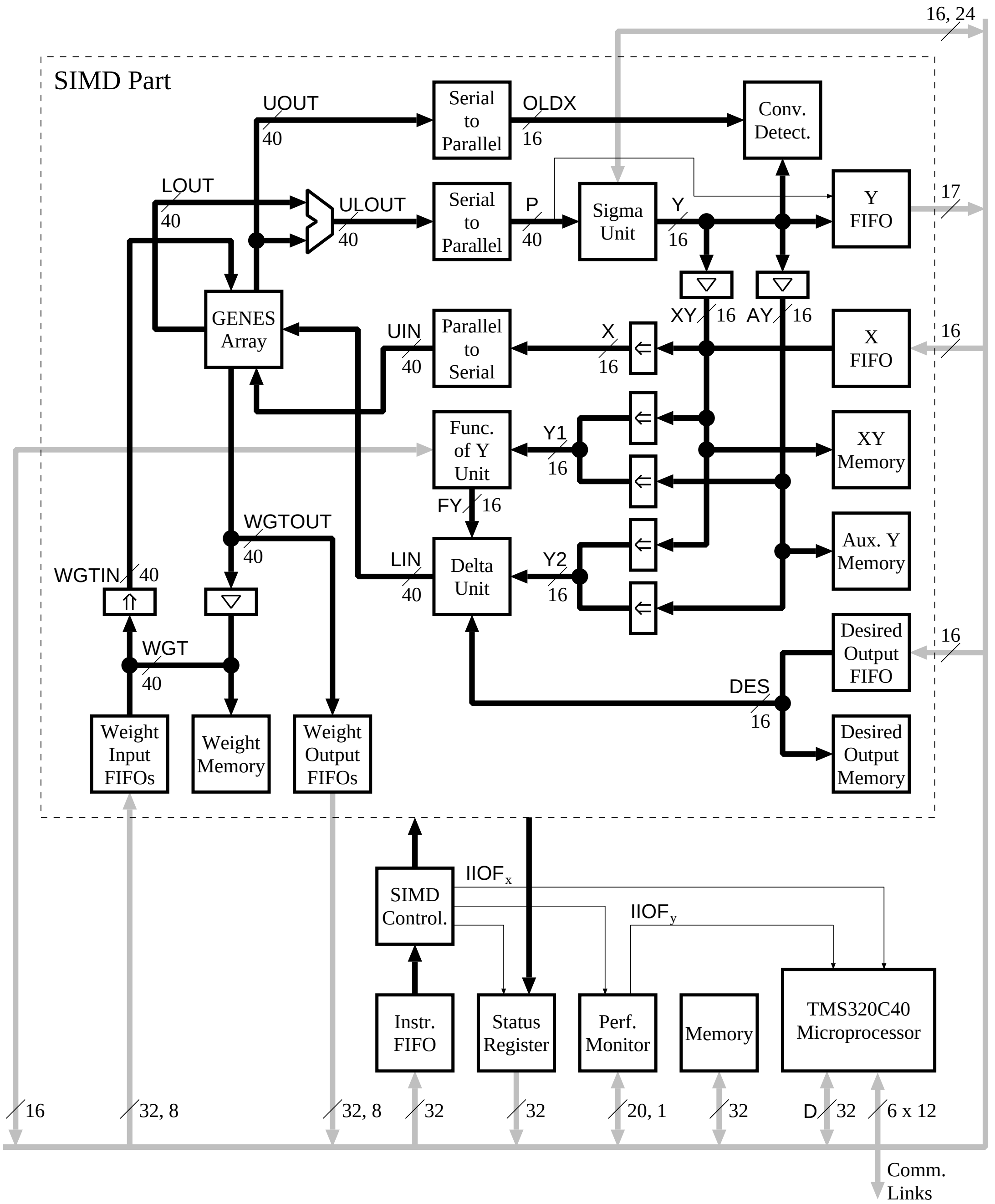

Figure 4: Architecture of the MANTRA I machine.

3.2 The non-linear function units

The *sigma unit* and the *function of Y unit* are used to implement arbitrary, usually non-linear, functions. In order to keep the hardware simple, they were implemented as look-up tables. Since the inputs and outputs of the function of Y unit are 16 bits wide, there was no problem to implement a full table. Four such tables can actually be prepared in memory and selected at run-time without stopping the SIMD part.

The design of the sigma unit was less simple, because it converts a 40-bit value into a 16-bit one. Since a table of 2^{40} entries (2 Tbyte) is obviously unrealistic, and since the truncation of the input bits would not yield an accurate enough result — at least for some parts of the function — another solution had to be found. The chosen implementation is

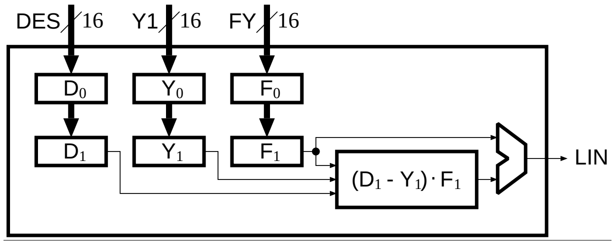

Figure 5 : Architecture of a delta unit's cell.

a double look-up table, consisting in a *coarse-grain table* mapping the whole input space at reduced precision, and a *fine-grain table* mapping a small window at high precision. A comparator allows to place this window anywhere in the input space (aligned on multiples of half of its size). A mechanism allows to zoom from full precision (one entry per input) to one entry for 128 inputs, hence trading the table's size with its precision.

3.3 The delta unit

Algorithms such as the delta or back-propagation rules, require the multiplication of the corresponding elements of two vectors, one of them being the difference of two vectors:

$$\delta_i \quad = \quad \varepsilon_i \cdot \sigma'(p_i) \quad = \quad (d_i - y_i) \cdot \sigma'(p_i) \tag{7}$$

This operation can not be efficiently performed by the 2-D GENES IV array, but can be carried out by the *delta unit*, an extra column of specialized cells. The inputs of these cells, shown in figure 5, are parallel while their outputs are serial. The cell's arithmetic unit can be bypassed when the computation of equation (7) is not required, the delta unit acts then as a parallel-to-serial converter. Three such cells are integrated on a VLSI chip, called *GACD1*, containing 16896 transistors (769 standard cells) on a die of $3.2 \times 3.2\,\mathrm{mm}^2$.

4 The control part

The control part of the MANTRA I machine, groups all the units shown outside the dashed box of figure 4. Its tasks are to configure the SIMD part, to dispatch instructions, and to manage the inputs and outputs. This part is a complete SISD system based on the TMS320C40 microprocessor from Texas Instruments running at 20 MHz. It should also handle the communications with a host computer and between different interconnected MANTRA I computers.

The main feature, that led to the choice of the TMS320C40 microprocessor, is the availability of six built-in communication channels (8 bits wide, 20 Mbyte/s). These links allow to connect multiple MANTRA I machines together and to a host computer. Other characteristics of the TMS320C40 taken into account are its two busses, its six built-in general-purpose DMA channels, and its 8 Kbyte of internal RAM.

4.1 The SIMD interface

As discussed in section 3.1, FIFOs are used to transfer data from the microprocessor to the SIMD part and vice-versa. Instructions are also written into the *instruction FIFO* by the microprocessor before being interpreted by the *SIMD controller*. If this FIFO becomes empty, the whole SIMD part is frozen until a new instruction is available.

The MANTRA I system follows the *very long instruction word (VLIW)* philosophy, that is, different fields of the instruction word are used to control the different units of the machine. Resource parallelism must therefore be explicitly controlled by the instruction flow. Instructions act on a macro-cycle basis, the SIMD controller being in charge, every clock cycle, of generating all the signals. A counter — one of the instruction's field — allows the same instruction to be repeated up to 128 times.

Besides the actual instructions, *auxiliary instructions* can also be sent through the instruction FIFO. They are either configuration information — such as which table should be used in the sigma or function of Y units — or initial addresses for the different memories. Except for the desired output memory, two addresses are attached to each memory, one being used for the data transmitted to the GENES IV array, and the other for the data received from it. These addresses are incremented after each access.

Since the interface between the SIMD and control parts consist only of asynchronous FIFOs and of a few double-synchronized status signals,[4] these two parts can run on two separate clocks.

4.2 The performance monitor

Performance degradation in the MANTRA I machine can be of two types: static and dynamic. The former type occurs when only a subset of the PEs is used during all or part of the computation. This is the case when the synaptic weight matrix — or sub-matrix if virtual networks are used, as described in section 2 — is smaller than the GENES IV array. Though, the penalty of filling and emptying the pipeline at the beginning and end of the computation belongs to this category, it is usually neglectable. Dynamic performance degradation, on the other hand, is due to stall cycles when instructions are lacking.

Given an algorithm and a set of parameters, the static performance of the MANTRA I computer can be calculated with simple formulas. There is therefore no need for a special hardware. The dynamic performance is however not deterministic, since it can be caused by delays in the communication between the MANTRA I system and a host computer, or any other systems providing inputs or requiring outputs. A dedicated unit, the *performance monitor*, was therefore implemented to compute the percentage of clock cycles when the SIMD part is active.

5 Heterogeneous-node network

As explained in section 4, a key feature of the TMS320C40 microprocessor is the availability of built-in communication channels, which allow the MANTRA I machine to be connected to a network of heterogeneous nodes, as shown in figure 6.

The first type of nodes a MANTRA I system can be connected to, is a host computer. An SBus board containing a TMS320C40 was developed to connect the system to any SUN 4 or SUN SPARCstation. It implements *direct virtual memory access (DVMA)*.

Another important element of this network, is a TMS320C40-based parallel processor, that can be used for any type of non-neural pre- or post-processing. Such a machine was developed at our laboratory [11]. In this system, individual processors are interconnected through cables. Though figure 6 shows a torus, any topology can be hard-wired: 1-D rings (or linear arrays) to 3-D hyper-tori (meshes), hypercubes up to six dimensions, etc. If the six communication links are used by the parallel processor (3-D hyper-tori or 6-D

[4] Figure 4 shows that the sigma and function of Y units are also directly interfaced to the microprocessor. However, these units are accessed (initialized) only when the SIMD part is inactive (frozen).

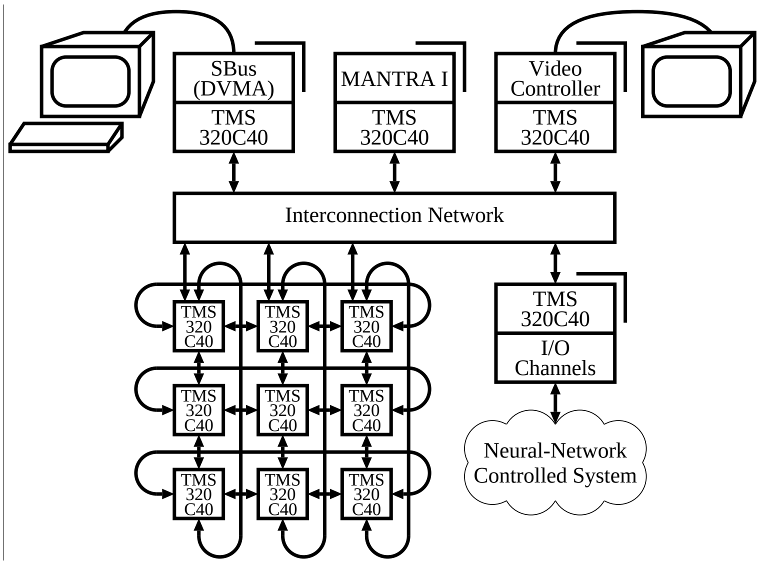

Figure 6 : Possible heterogeneous network hosting MANTRA I machines.

hypercubes), than the topology must be slightly altered to implement a connection to other nodes of the heterogeneous-node network.

Any other dedicated node, such as a video-controller or an interface to a real-time system to be controlled, can easily be implemented as an extension to the TMS320C40 processor board of the presented system.[5]

6 Project status

The MANTRA I machine is built of four different boards : a general-purpose processor board and three dedicated boards shown in figure 7 :

1. **Processor board :** This board was designed by C. Marguerat [11] as a building block for a parallel processor. It contains 33 chips.
2. **Control board :** This board groups all the control logic and the storage units of figure 4 (138 chips).
3. **Input/output board :** This board contains the delta, parallel-to-serial, and serial-to-parallel units of figure 4. It is made of 465 chips including 14 GACD1 chips.
4. **GENES IV array board :** This board houses a matrix of 100 GENES IV chips. One such board is required for small configurations (up to $20 \times 20 = 400$ PEs), and four boards for larger ones (up to $40 \times 40 = 1600$ PEs),

As of June 1993, a first prototype of the machine has been designed, manufactured, and successfully debugged. It is running on a scaled-down clock of 8 MHz (instead of 10 MHz). A second revision of the control board is currently under manufacturing. It should eliminate the electrical problems (ground bounce and reflections) encountered on the first version and allow the machine to run at its nominal clock.

A first algorithm, the delta rule, has already been ported. It showed that dynamic utilization rates of 97 % to 99 % can easily be achieved when the data are present in

[5]The MANTRA I computer is actually implemented as such an extension.

(a)

(b)

(c)

Figure 7: MANTRA I printed circuit boards. (a) Control board. (b) Input/output board. (c) GENES IV array board. (Photographs: Heidi Viredaz-Bader.)

the microprocessor's memory. Other algorithms (back-propagation rule, Kohonen model, etc.) as well as a communication library are currently under development.

7 Conclusion

The GENES IV architecture, presented here, is a good candidate for an SIMD processor array dedicated to neural computation. Its strength comes from its large number of simple PEs that can sustain a utilization rate close to 100 %. Its good scalability, makes it well suited for WSI.

This systolic array has been integrated in the MANTRA I neural computer. Its design was mainly focussed on sustaining the instruction flow and input/output data flows required to keep the GENES IV array busy.

Due to its heavily pipelined nature, the MANTRA I machine suffers from the typical restrictions of large pipelines. That is, the targeted applications must present an intrinsic parallelism, that is large enough to hide the pipeline latency — which has, in the framework of neural computation, for consequence that applications should be insensitive to the use of large epochs — and the problems must be sufficiently large.

Acknowledgements

The author would like to thank Prof. Jean-Daniel Nicoud, the laboratory director, without whom this project would not have been possible. The work of François Blayo and Christian Lehmann provided the bases of the present project. The author deeply acknowledges the help of Paolo Ienne who co-designed the architecture of the GENES IV and GACD1 chips, and whose careful work at implementing and simulating these chips led to a first working silicon in both cases. Special thanks go to Christophe Marguerat for his advise with the TMS320C40, and whose processor board proved to be of great help in testing the MANTRA I machine. Finally, Georges Vaucher and Peter Brühlmeier should be thanked for their outstanding work at placing and routing the different printed circuit boards, and André Guignard for his help in packaging the VLSI circuits.

A Summary of neural algorithms

A.1 The back-propagation rule

The following equations describe the back-propagation rule using epoch updating for an L-layer feed-forward network. Equations 8 to 10 are evaluated for each layer q and for each vector $\vec{\mathbf{x}}\,(s)$ of the current epoch : $s_1 \leq s \leq s_2$. The synaptic weights $\mathbf{W}^{[q]}$ are then updated according to equation 11. This process is repeated for each epoch.

$$
\begin{aligned}
\vec{\mathbf{y}}^{[q]}(s) &= \sigma\left(\vec{\mathbf{p}}^{[q]}(s)\right) = \sigma\left(\mathbf{W}^{[q]} \cdot \vec{\mathbf{y}}^{[q-1]}(s)\right) && \text{for : } q = 1, 2, \ldots, L && (8)\\
\delta_i^{[L]}(s) &= \left(d_i(s) - y_i^{[L]}(s)\right) \cdot \sigma'\left(p_i^{[L]}(s)\right) && && (9)\\
\delta_i^{[q]}(s) &= \vec{\mathbf{W}}_i^{[q+1]\mathrm{T}} \cdot \vec{\boldsymbol{\delta}}^{[q+1]}(s) \cdot \sigma'\left(p_i^{[q]}(s)\right) && \text{for : } q = 1, 2, \ldots, L-1 && (10)\\
\mathbf{W}^{[q]} &:= \mathbf{W}^{[q]} + \alpha \cdot \sum_{s=s_1}^{s_2} \vec{\boldsymbol{\delta}}^{[q]}(s) \cdot \vec{\mathbf{y}}^{[q-1]\mathrm{T}}(s) && \text{for : } q = 1, 2, \ldots, L && (11)
\end{aligned}
$$

Where the vector $\vec{\mathbf{W}}_i^{[q]}$ is the i^{th} column of the matrix $\mathbf{W}^{[q]}$, and $\vec{\mathbf{y}}^{[0]}(s) = \vec{\mathbf{x}}\,(s)$.

A.2 The Kohonen model

The following equations describe the Kohonen model, using the Euclidean distance, winner take-all, and epoch updating.

$$y_i(s) = \sqrt{\sum_{j=1}^{N} \left(x_j(s) - W_{i,j}\right)^2} \tag{12}$$

$$y_{I(s)}(s) = \min_i \left(y_i(s)\right) \tag{13}$$

$$\mathbf{W}_i := \mathbf{W}_i + \alpha \cdot \sum_{\substack{s=s_1 \\ \forall s:\, i \in N(I(s))}}^{s_2} \left(\vec{\mathbf{x}}^{\mathrm{T}}(s) - \mathbf{W}_i\right) \tag{14}$$

Where the vector $\mathbf{W}_i$ is the i^{th} row of the matrix $\mathbf{W}$, and $N(i)$ is the set of neighbors of i. Neuron $I(s)$ is usually called the winner (for the input s).

References

[1] F. Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review*, 65:386–408, 1958.

[2] Bernard Widrow and Marcian E. Hoff. *Adaptive Switching Circuits.* In *IRE-WESCON Convention Record*, pages 96–104, New-York, USA, 1960.

[3] Yann Le Cun. *A Learning Scheme for Asymmetric Threshold Network.* In *Proceedings of Cognitiva 85*, Paris, France, June 1985.

[4] John J. Hopfield. *Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Sciences*, 79:2254–2258, April 1982.

[5] Teuvo Kohonen. *Analysis of a Simple Self-Organizing Process. Biological cybernetics*, 44:135–140, 1982.

[6] Dagmar Niebur and Alain J. Germond. *Unsupervised Neural Net Classification of Power System Static Security States. International Journal on Electrical Power and Energy Systems*, 114(2 & 3):233–242, April & June 1992.

[7] François Blayo. *Une Implantation Systolique des Algorithmes Connexionnistes.* Ph.D. thesis N° 904, EPFL, Lausanne, Switzerland, 1990.

[8] Christian Lehmann, Marc Viredaz, and François Blayo. *A Generic Systolic Array Building Block for Neural Networks with On-Chip Learning. IEEE Transactions on Neural Networks*, 4(3), May 1993. To appear.

[9] Christian Lehmann and François Blayo. *A VLSI Implementation of a Generic Systolic Synaptic Building Block for Neural Networks.* In José G. Delgado-Frias and William R. Moore, editors, *VLSI for Artificial Intelligence and Neural Networks*, chapter 4.8, pages 325–334. Plenum Press, 1991.

[10] Marc A. Viredaz, Christian Lehmann, François Blayo, and Paolo Ienne. *MANTRA: A Multi-Model Neural-Network Computer.* In *Proceedings of the 3rd International Workshop on VLSI for Neural Networks and Artificial Intelligence*, Oxford, UK, September 1992.

[11] Christophe Marguerat. *Artificial Neural Network Algorithms on a Parallel DSP System.* Internal Report no. R93.23C, LAMI, EPFL, Lausanne, Switzerland, 1993.