ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Swiss Federal Institute of Technology, Lausanne — Microcomputing Laboratory

Architectures for Neuro-Computers: Review and Performance Evaluation

Paolo Ienne

Technical Report no. 93/21
January 1993

Abstract

As the field of neural-network matures toward real-world applications, a need for hardware systems to efficiently compute larger network arises. Several designs have been proposed in the recent years and a selection of the more interesting VLSI digital realizations are reviewed here. Limitations of conventional performance measurements are briefly discussed and a different architectural level evaluation approach is attempted by proposing a number of characteristic performance indexes on idealized architecture classes. As a results of this analysis, some conclusions on the advantages and limitations of the different architectures and on the feasibility of the proposed approach are drawn. Architectural aspects that require further developments are also emphasized.

Architectures for Neuro-Computers: Review and Performance Evaluation

Paolo Ienne

January 1993

Abstract

As the field of neural-network matures toward real-world applications, a need for hardware systems to efficiently compute larger network arises. Several designs have been proposed in the recent years and a selection of the more interesting VLSI digital realizations are reviewed here. Limitations of conventional performance measurements are briefly discussed and a different architectural level evaluation approach is attempted by proposing a number of characteristic performance indexes on idealized architecture classes. As a results of this analysis, some conclusions on the advantages and limitations of the different architectures and on the feasibility of the proposed approach are drawn. Architectural aspects that require further developments are also emphasized.

Contents

1 Introduction

The field of *Neural-Networks (NN)* is in many cases extending above the typical boundaries of academic research. Applications to real-life problems tend to become competitive with well-established traditional techniques. In many cases, the comparisons of traditional, well-mastered techniques to neural networks experiments appear fragmentary or anecdotal due to the lack of significative benchmarks or real-world applications. Recent experiences (as the Santa Fe Institute competition on *Time Series Prediction and Analysis* whose results are described in [Wei93]) show the maturity of some NN algorithms and the importance of their intrinsic non-linearity contrasted to classical linear approaches. Weigend and Gershenfeld affirm:

> *Quite to our surprise, it turned out that connectionist networks consistently outperformed all other methods received, ranging from visual predictions to sophisticated noise reduction techniques.*

Unfortunately, real applications tend to require large networks and/or large data sets for the learning phase. This makes implementation on ordinary hardware platforms extremely inefficient. Considering again [Wei93], it can be seen that the four best connectionist entries took between three hours to three weeks to run, the quickest one being programmed on a CRAY Y-MP. Difficulties in experimenting with large networks also hinder researchers from exploring the field. It should be considered that often a good result is obtained by repeatedly learning the problem prototypes with different parameters to search for an optimal tuning, this empirical but nevertheless necessary process becoming prohibitive for sufficiently large networks.

Given the growing practical interest of NN, the research community is presenting a growing attention to hardware platforms where connectionist algorithms can be run efficiently and cost-effectively. If commercial implementations are still ahead in the future (with few exceptions such as the Adaptive Solution or the HNC systems), most of the important hardware manufacturers are active in the field: Philips, Siemens, Hitachi, Intel, AT&T to mention some.

This document focusses on digital-electronic accelerators, typically—but not exclusively—in the form of machines designed around some application specific integrated circuit. The purpose is to establish some quantitative criteria to evaluate and compare the different NN accelerator architectures proposed in literature. While the details of this evaluation will be addressed in the sequel, the idea of accelerator is best precised since now. An accelerator should have the following characteristics:

- An accelerator is not a stand-alone machine. It is usually connected with some traditional computer such as a scientific workstation or a personal computer. The host machine runs the user interface and all the non-connectionist software. The accelerator is conceived to run efficiently neural algorithms, possibly exploiting at best the intrinsic parallelism they present. When the host has one of these tasks to run, it sends the input data to the accelerator with the necessary information on the algorithm; at the end of the computation, it gets the output data from the accelerator.

- An accelerator should typically be a multi-model machine, that is the user should be able to map on it most of the known connectionist algorithms. This may not always be the case, some machines being dedicated to widely used algorithms as the back-propagation.

- The size and shape of the network simulated on the accelerator must be reconfigurable, allowing user-defined problems to be freely mapped on the hardware.

Sometimes, the term *neuro-computer* will be used instead of accelerator, often referring to large, highly versatile machines.

The purpose of the rest of this introduction is to explicit some view-points that characterise the whole analysis and to clarify their reasons. The choice of limiting to the digital technology is discussed in §1.1. The reasons for it are also connected with a number of basic assumptions on the ideal characteristics demanded to a NN accelerator useful for complex practical applications and for research purposes. These assumptions and the key features for an accelerator architecture to be apt for a future evolution are discussed in §1.2. Once the basic hypotheses have been discussed and justified, the task of obtaining performance measurements is briefly addressed and the difficulties acknowledged in §1.3. Given the mentioned problems, the meaning of a performance evaluation for the purpose of the present document is precised in §1.4. The structure of this document is presented in §1.5 while the notation used throughout the text is defined in §1.6.

1.1 Digital Vs. Analog

The first choice to be addressed when considering electronic implementations of neural networks is the selection of the analog or digital approach. In a sense, the analog approach is more promising: Nature seems to replicate a few classes of simple *devices* an enormous number of times and allocates each *computation* to a different neuron (i.e., there seems to be no biological mechanism equivalent to data-multiplexing). This somewhat corresponds to analog processing elements, usually composed of only a few transistors, each element representing either a synapse or a neuron. On the contrary, current digital implementations are complex, have several thousands of transistors (e.g., GENES IV—described in §2.3.1—has approximately 18000 transistors per synapse, N64000—described in §2.1.1—has some 175000 per neuron) and almost always present some sort of data-multiplexing to outweight the processing element complexity. Still, there are other elements making digital implementations interesting:

- **Precision.** Analog implementations hardly reach more than a few bits of precision. The attained precisions are typically insufficient for learning algorithms as the back-propagation rule, which seem to be very sensitive on the resolution. (Experimental determinations of precision requirements in [Asa90] indicate that a minimum of 16-bit precision on the weights is required for back-propagation networks.)

- **Programmability** and **flexibility.** The field of neural network is still young: New algorithms are presented almost continuously and the originating biologic models are still far away both in network complexity and in algorithmic description. Researchers requiring accelerators need to implement and study the behaviour of novel networks or learning rules. The required degree of freedom is almost impossible to attain in analog implementations: These are usually strictly dedicated to a single model (and make use of technologic peculiarities to implement it) and thus are scarcely usable in a general neuro-computer.

Probably, both approaches have an interesting future, depending both on the application (on-the-field application-specific hardware or desk-top accelerators for research purposes) and on the processing stage (sensor outputs preprocessing or final data processing). This document, as explained in the next section, tends to concentrate on high computation-power multi-model accelerators. These conditions, together with the general maturity of the digital technology, make it the current best choice for the type of application considered.

1.2 Features of a Multi-Model Accelerator

As mentioned, the purpose of the present research is to compare the different architectures described in literature. Any analysis critically depends on the basic criteria it is built on. The following list mentions the key features that are assumed as fundamental for an architecture to be well-adapted for a neuro-computer and, at the same time, to be open to future developments:

- **Scalable.** The conception of an architecture should consider as a fundamental fact the technologic trend and should thus be conceived to easily take the best advantage of higher density of integration. An higher density, in this field, directly implies the possibility of implementing more and more processing elements on the same die and, consequently, in a system.

- **Programmable.** The issue has already been addressed when the digital/analog debate has been addressed (see §1.1). A good architecture should allow the reconfiguration at software level of existing hardware to make the most of the physically available processing elements. Moreover, this should be possible without requiring the user to have a complete understanding of the implementation details of the machine (as a C-programmer of a workstation can be totally unaware of the complexity of the instruction set of the CPU).

- **Modular.** Neuro-computers tend to differentiate from common massively parallel super-computers in their specificity to a particular class of problems. It is at all possible that some new algorithms may require new operators and data flows hardly obtainable simply by making the machine *programmable* in the sense mentioned in the previous bullet. *Modularity* indicates the possibility of creation, under a common framework, of compatible new elements required by the novel networks without having to redesign the whole machine.

- **Virtual Networks Support.** Any neuro-computer is composed by a number of modules (often corresponding to synapses or neurons) that allow an optimal implementation of networks up to a determined maximum size. Real problems are often large and seldom have a physical implementation available which is large enough. The usefulness of a given architecture strongly depends on the possibility of implementing networks larger than those suggested by the physical size of the machine. For instance, in a linear array of N non-pipelined multipliers, typically conceived for being assigned a single neuron each, it is important to be able to map also networks with more than N neurons. It is evident that—when this is the case—part of the parallelism is necessarily lost and the network has to be partitioned and the sections processed in a serial way, thus decreasing the throughput. Unfortunately—as examples in §2 show—this possibility is not always available or, when it is, the performance may drops considerably.

Performance assessment criteria should reflect these features.

1.3 Traditional Performance Evaluation and Measurement

It can be instructive to compare performance measurements of neuro-computers with the corresponding activity for traditional computers or super-computers. The field of computer performance assessment, despite some aspects still controversial, is well established and exploited both by the product marketing to promote new products and by purchasers to select the optimal machine for their application. Unfortunately, there is one important difference between ordinary computers and neuro-computers: While the former systems are commercially available machines, easily attainable for a test and with widely available and portable software, the latter ones are not. VLSI circuits for neural network emulation in hardware are working experimental devices in best cases, more often being reported in the literature as proposed architectures undergoing implementation processes at the time of writing. Furthermore, the relative youngness of the field implies the absence of widely acknowledged application *software* (in the same sense that GCC, TEX or Spice can be considered typical application programs for a work-station or a personal computer, see [HEN90]). Only few exceptions exist, as the NETtalk network ([SEJ87]), and they are sometimes used for comparisons. Due to this problem, no serious effort seems to have been made until now to develop a comprehensive benchmark suite for NN accelerators. (The Carnegie Mellon University maintains a benchmark suite but—besides not being sufficiently acknowledged to be used for comparisons—it seems more targeted to algorithm convergence evaluation than to hardware efficiency quantification.)

The traditional approach for quantifying NN accelerator performance is to measure the number of multiply and accumulate operations performed in the unit time (measured in *MCPS* or *Millions of Connections Per Second*) and the rate of weight updates (measured in *MCUPS* or *Millions of Connection Updates Per Second*). These two measures somewhat correspond to the MIPS or the MFLOPS measured on traditional systems. It is usual for any system description in literature to give this sort of measurement. A first problem to be mentioned is the fact that no standard network (topology, prototypes, final error, etc.) exists to perform the calculation (apart from the mentioned exception of NETtalk), thus allowing each designer to choose a best-case network and producing a result that misses any information on the capability of the structure to adapt to different problems. But there is more than that: the MCPS and MCUPS definition is extremely vague if not simply unexisting, even to levels well exceedings the problems of MIPS or MFLOPS discussed by [HEN90]. Two extreme examples show clearly the problem. In [McC91], an example of mapping of one of the largest possible back-propagation networks on the ADAPTIVE SOLUTION architecture is described. A calculation of the performance in MCPS and MCUPS is performed by ignoring external effects limiting the system performance, such as the external communication problems. A fair *peak* MCUPS performance is thus obtained as the product of the number of connections in the network by the number of input vectors completely processed in the unit time. At the opposite of this computation style are the reports on the PHILIPS Lneuro 1.0 circuit which give the following results:

	Peak MCPS	Peak MCUPS
PHILIPS ([THE90])	26	32
ADAPTIVE SOLUTIONS ([McC91])	9,671	2,379

(the ADAPTIVE SOLUTIONS data have been added for comparison). Most readers would probably be very surprised to see that the MCUPS are higher than the MCPS. One could wonder how the update phase can be quicker than the forward phase when the former one algorithmically requires a full forward and a back-propagation phase for the update to be possible! The problem results from a misunderstanding on the MCUPS meaning, intended by the authors of the Lneuro as the time to perform the operation $\mathbf{W}^{\mathrm{NEW}} = \mathbf{W}^{\mathrm{OLD}} + \vec{\delta} \cdot \vec{\mathbf{x}}^{\mathrm{T}}$ and leaving out the time to perform the $\vec{\delta}$ computation. Apart from disqualifying the MCPS/MCUPS system as it is used today, this problem raises another: How can this type of measure be performed on hardware relying on the host for some basic operation required for the network implementation? The Lneuro actually misses the term-to-term multiplication of vectors required for the weight back-propagation, thus making a true peak MCUPS rate host dependent.

These difficulties explain the more qualitative approaches usually taken by other authors in comparing different designs (e.g., see [ALI92] or [GUE92]).

1.4 Rationale of Architectural Evaluation

An important remark is that the object of this document is to compare different architectures rather than implementations (even if these exist). The idea is that commercial products (such as ordinary CPUs) have a rather homogeneous technologic level, because all the manufacturers are interested in obtaining the best placement in the market for their architecture. This is not at all the case for experimental NN devices where the authors are more interested in showing the feasibility of an approach than in producing the best implementation (e.g., the PHILIPS Lneuro is implemented with 1.6 μm standard cells, [MAU92]). Economics and environment also play an important role: Academic projects such as GENES IV ([VIR92]) may have limited fundings and also choose to go for quick (and relatively inefficient) implementation techniques as standard cells. Large companies research projects, such as the Hitachi's one ([YAS90]), may have readily available advanced technologies, e.g., 0.8 μm CMOS at Wafer Scale Integration (WSI). Similarly, commercial projects (e.g., [HAM91]) may decide to heavily invest in advanced, but yet consolidated, technologies as a key factor to reach interesting performances. Thus, heterogeneity imposes to concentrate the attention on architectural differences, ignoring—as much as possible—implementation details.

A similar approach was taken in 1976 by the Army/Navy *Computer Family Architecture (CFA)* Committee in selecting an optimal computer architecture for future military implementations ([FUL77]). The idea was that, the selected architecture undergoing a novel implementation process, the original implementation details were totally irrelevant and the evaluation to be taken independently. The CFA developed—on the period of a couple of years—a system of quantitative performance indexes to measure specific characteristics of the candidates (filtered after a set of absolute criteria). These performance indexes, combined in a single weighted value, where used to rank the candidates and select three architectures on which to implement rather comprehensive benchmarks.

The present work is remotely inspired to that experience, all the differences in context taken into account.

1.5 Structure of the Document

This document articulates essentially in two sections, apart from the present introduction. The first (§2) reviews a number of architectures under a first tentative grouping of similar systems. The description of the designs is generally very short, concentrating rather on a non-systematic analysis of their most interesting aspects or of the inherent limitations.

In §3 a more systematic analysis is attempted by substituting the implementation with idealized structures. The architecture classes are obtained after filtering out some candidates after a set of absolute criteria. These criteria are not conceived to screen out the architectures but rather to resume some fundamental aspects of the designs. These help to gain a comparative view of the elements presented in the first section. Once the idealized architectures have been presented, a set of quantitative performance indexes are defined and evaluated on the examples. The results are discussed and some conclusions drawn on the approach.

The document concludes with some more general considerations on problems connected with the design of digital neuro-computers. Appendix A contains the details of the computations of the performance indexes.

1.6 Notation

In the present document the following notations have been adopted: matrixes and vectors are written in bold characters (e.g., $\mathbf{M}$), while scalars are printed in italic (e.g., v). In addition, vectors (which can be regarded as single-column matrixes) are shown with an upper arrow (e.g., $\vec{\mathbf{v}}$). Subscripts are used to denote rows of matrixes (e.g., $\mathbf{M}_i$) and elements of matrixes and vectors (e.g., $M_{i,j}$ and v_i).

Multilayer network will often be characterised as, for instance, a *(3-4-5-6) 3-layer network*. This means that the network has three inputs, four neurons in the first hidden layer, five in the subsequent and six output neurons.

Appendix B describes the simplest version of the back-propagation algorithm, often referenced in the text, and defines the symbols used in the document.

2 Neuro-Computer Architectures

The present section is devoted to the presentation of examples of architectures and devices described in literature. The domain has not been restricted to realizations strictly conforming to the features described in §1.2 or to real multi-model machines. In fact, only very few existing architectures could claim to be so and others show some interesting features. The purpose of this section is to review representative architectures to develop a feel for characteristic quantities that will be used in the sequel to produce quantitative selection criteria (see §3). Moreover, some of the data-flow structures actually may support other operating mode which have not been implemented by the authors, perhaps because of slightly different research goals.

2.1 Generic Parallel Architectures

This section presents the non-systolic architectures developed to support NN algorithms. They differentiate from systolic architectures in the fact that no part of the computation for a new vector can be started before the previous is done.

2.1.1 Adaptive Solutions' CNAPS

The Adaptive Solutions's architecture, named CNAPS, is described in several papers dealing with different aspects of the implementation. A basic description of the structure can be found in [Ham91] while more details on the VLSI implementation (inova N64000 Digital Neural Network Processor) are given in [Gri91] and [Ino]. Interestingly enough, this is almost the sole commercial neuro-computer marketed so far. An informal Adaptive Solutions communication ([Hay92]) reports 30 machines installed so far and fully functional. Current list prices range from 62,500 USD to 137,500 USD, depending on system configuration. The implementation itself is extremely interesting: An aggressive design, the die measures about 1 square inch and has 13.6 millions of transistors fabricated. The chip contains 64 *Processing Nodes* (PNs). A three-level redundancy (raising the number of fabricated PN to 80) should guarantee a 90% yield. Active transistors are thus reduced to 11 millions.

The basic architecture consists of a number of PNs, constituting a *Single Instruction stream, Multiple Data stream* (SIMD) 1D array. The PN internal architecture is a compromise between the conception of a general processor and neural network dedicated unit. The authors claim that all the neural algorithms they examined can be mapped on the structure, as well as other digital signal processing, pattern recognition and rule processing (fuzzy) applications. As mentioned, each single chip contains 64 PNs, connected in the 1D array that extends throughout the whole machine.

Figure 1 shows the connections between PNs on different chips and the sequencer. It can be observed the reduced connectivity: essentially, only two 8-bit and one 31-bit busses connect the chips among them and with the sequencer. The figure suggests the two bigger advantages of the architecture:

- The regularity of the structure, allowing expansion of the machine by simply adding more N64000 on the bus (the number of chips accounting for the configuration differences mentioned above in the pricing details).
- The minimal connectivity, making the processor feasible and reducing packaging and mounting costs.

The key to the low connectivity is shown in figure 2: Each PN contains 4 Kbytes of static

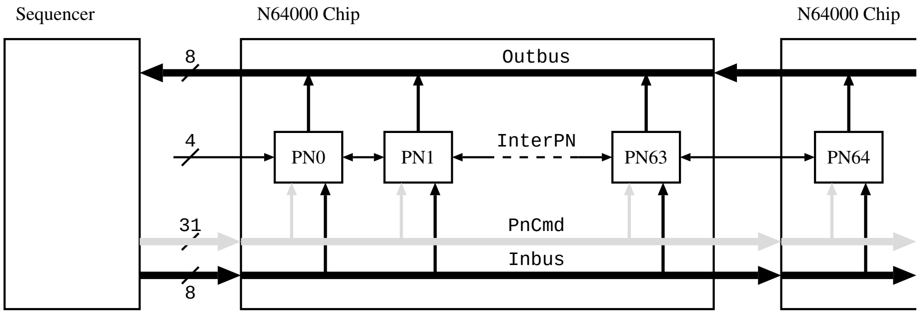

Figure 1. CNAPS. Inter-chip Communication.

RAM to hold the weights. To examine what this implies, lets consider the classic matrix/vector multiplication $\vec{\mathbf{p}} = \mathbf{W} \times \vec{\mathbf{x}}$, where $\mathbf{W}$ has size $N \times M$ and $\vec{\mathbf{x}}$ is composed by M elements. Supposing that the number of PN is greater or equal to N and that M is smaller than 4096, the matrix $\mathbf{W}$ can be completely stored in the memory of the PNs, one row in each PN. At this point it is simply needed to present one element of $\vec{\mathbf{x}}$ at each cycle on the parallel input bus of the array: Each PN multiplies every input by one of its weights, in succession, and accumulates the partial result. When the M elements of $\vec{\mathbf{x}}$ have been presented, each of the N active PNs contains the corresponding value of the output $\vec{\mathbf{p}}$. If $\vec{\mathbf{p}}$ is the final result, N cycles are needed to extract the vector on the single output bus. If, on the contrary $\vec{\mathbf{p}}$ is being reused (after application of the non-linear function $\sigma(\bullet)$, implemented as a table look-up in the weight memory), the elements can be output one per cycle and sent immediately to the input bus for the next processing step.

This approach is followed in implementing the forward phase of the back-propagation algorithm on a system composed of 512 PNs ([McC91], [Bak91]). The error back-propagation, which requires the multiplication of the transpose of the weight matrix by the errors on the output of the layer, poses some more problems. The described solution implies the duplication of the weight matrix in the processors, once stored one row in each processor and once one column each. Both matrixes are updated. Neither paper describes how the simultaneous adaptation is performed, but it seems implied than the update process is simply repeated twice for each vector and for each layer, once for each matrix. The reported performance is quite astonishing: 2,379 MCUPS and 9,671 MCPS with 16-bit weights. This is calculated on a network with 1900 inputs, 500 hidden units and 12 outputs.

Of course, the low connectivity does not come for free: As explained, all the architecture is based on the PN stored weights. When the network size (measured in terms of size of the weight matrixes, possibly duplicated if needed in the back-propagation process, plus some space taken by the non-linear function look-up table) exceeds the capacities of the PN memory, time should be allowed to up-load the PNs with the weights. As there is an unique input bus, these can be fed at a rate of only one per cycle. To gain a better understanding of the loss in performance, the calculation of the test network of [McC91] can be repeated for the double of inputs[1]. Supposing all the cycles per node or per input element remain constant, and assuming—as it is

[1] It should be noted that the algorithm details in [McC91] are not sufficient for a precise calculation. For instance, the meaning and dependency of the *Overhead clock cycles per input vector* is not explained, nor is the exact function performed in each of the *Cycles per output* or *per hidden node*. The shown calculation may reveal more qualitative than really quantitative.

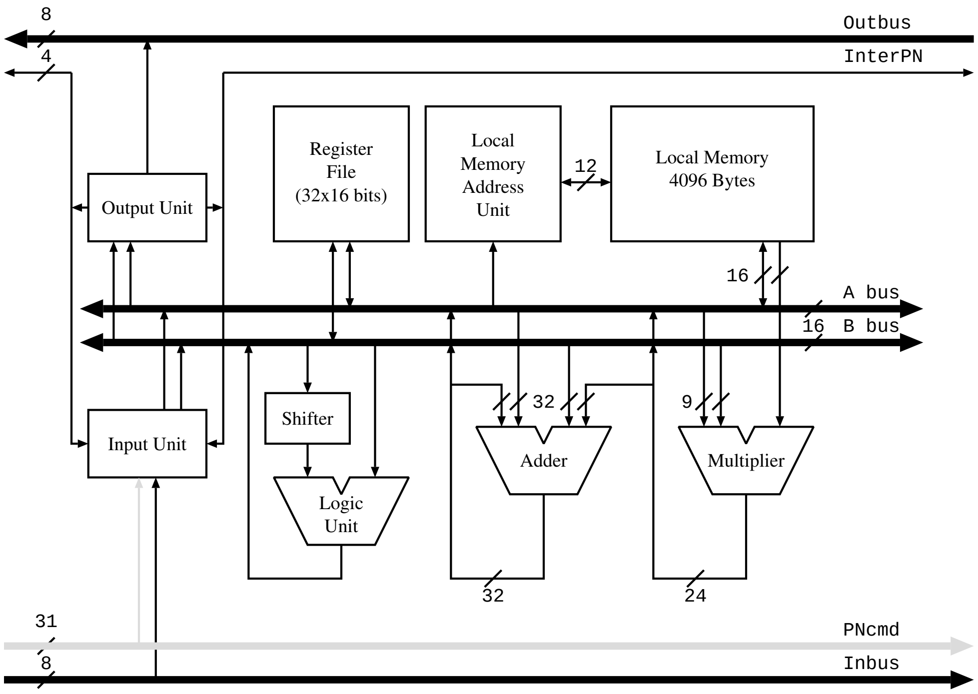

Figure 2. CNAPS. Block Diagram of a N64000 Processing Node.

almost the case—that the (1900-500-12) network already occupied the whole memory space, the performance[2] would be reduced to less than 25 MCUPS, or 1% of the previous value. In fact, what is happening is that the machine performs a bit worse than a single multiplier—connected with a sequencer and external RAM feeding data at a rate of 25 MHz. Of course, this is an extreme example: techniques as epoch processing and updating may reduce the impact of the swapping by performing it only once every n vectors.

In synthesis it can be affirmed that the main limit of the Adaptive Solutions's architecture is the strong inefficiency in case of virtual networks exceeding the memory capacity of the PNs. If the 4 Kbytes size is sufficiently generous for most of todays applications, it should be noted that complex multi-layer multi-model networks may require storage for a number of interconnections in the order of 10^5 to 10^{10} (see [DAR88, Figure 28.5]): the largest machine offered by Adaptive Solutions has 2 Mbytes available, that reduce approximately to 500 Kbytes in case of 16-bit weights, for a back-propagation algorithm[3].

[2] In detail:

- $Calculation\ cycles\ per\ vector = 360 + 3,800 \cdot 4 + 500 \cdot 4 + 12 \cdot 7 = 17,644$
- $Weight\ swapping\ cycles\ per\ vector = 3,800 \cdot 500 = 1,900,000$
- $Vectors\ per\ second = 25,000,000/(1,900,000 + 17,644) = 13$
- $Total\ forward\ weights = 3,800 \cdot 500 + 500 \cdot 12 = 1,906,000$
- $Weight\ updates\ per\ second = 1,906,000 \cdot 13 = 24,778,000$

[3] The (1900-500-12) test network, with its very high number of inputs and very low output count, is rather misleading. The duplicated weight matrix is actually very small (6000 elements) compared to the huge first layer matrix (950000 elements), thus reducing the duplication loss impact. Moreover the inclusion of a momentum in

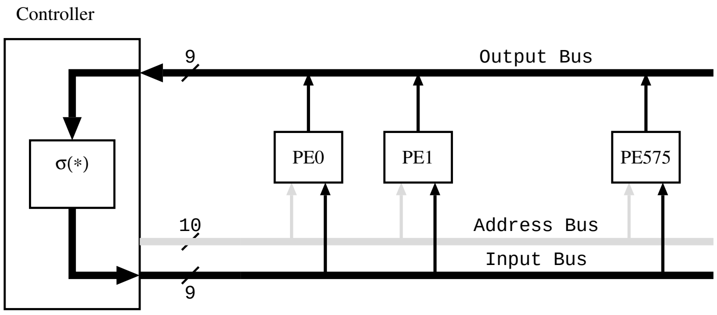

Figure 3. HITACHI. Time Sharing Bus Architecture.

On the other side, the great advantage of the CNAPS architecture is the versatility of the processing elements and their easy programmability, which allows other algorithms to be efficiently mapped on the machine (e.g., the Kohonen feature map described in [HAM] or image processing algorithms and Fourier transforms in [BAK91]). This makes the ADAPTIVE SOLUTIONS's machine very powerful when the memory limit is not reached.

2.1.2 HITACHI's WSI Networks

Two HITACHI machines will be considered in the sequel. The first design performs only a weighted sum of the neuron inputs, relying on independent hardware (e.g. traditional sequential host computer) to execute the learning algorithm. The second machine—directly issued from the first—implements also the back-propagation algorithm.

HITACHI's Feedforward-Only Network The first HITACHI's machine ([YAS90]) belongs essentially to the same broadcast bus cathegory of the ADAPTIVE SOLUTIONS' one (see figure 3). The address bus selects a weight from a register file in each of the processing element. The processing elements are greatly simpler than those implemented in the N64000 (see figure 4 for a detail of their inner structure). The operation performed here is only a weighted sum (8-bit coefficients) of the 9-bit inputs. The actual function implemented is $Output_{t+1} = c \cdot Output_t + (1-c) \cdot \sum_{i=0}^{N-1} W_i \cdot Input_i$. One peculiarity of the implementation is the small weight register file (64 entries compared to 576 neurons per wafer). The 64 locations are assigned (through the address part of the file) to the heaviest weights, the others being considered zero. During the output computation, the address bus is inspected, compared to each of the stored addresses and the weight, if found, multiplied by the value on the input bus and accumulated. The impact of this algorithmic simplification is not addressed in the cited article. The architecture cannot support virtual networks, nor NN paradigms other than the one described and it does not have support for on-chip learning rules. It is thus moderately interesting for a neuro-accelerator.

On the whole, the most attracting feature of the HITACHI machine is the physical realization: It is a *Wafer Scale Integration (WSI)* project integrating 576 processing elements on a 5" wafer. As in the case of the ADAPTIVE SOLUTION system, the reconfiguration is made relatively easy by the type of inter-processor connection network. Bad processors can simply be excluded by the linear array. In the HITACHI design, to improve yield and minimize the impact of defects

the algorithm would probably require a further duplication of the matrixes to store the $\Delta\mathbf{W}$.

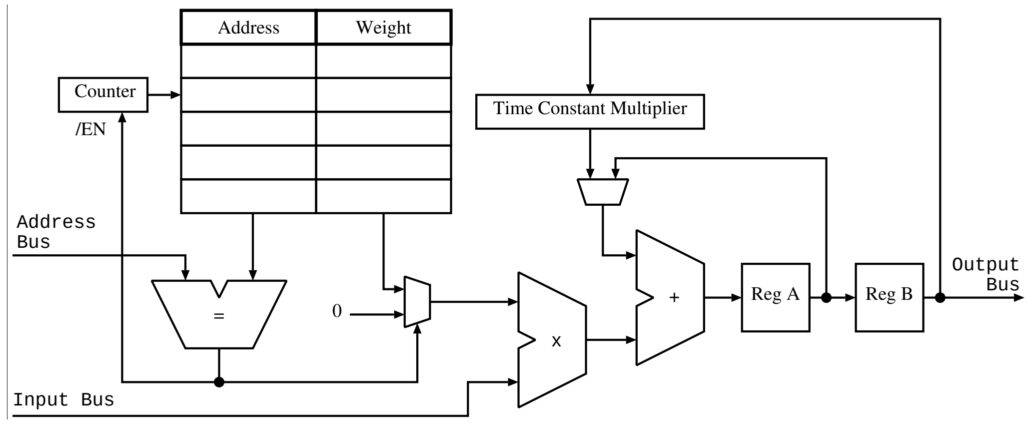

Figure 4. HITACHI. Processing Element Structure.

on internal busses, a hierarchical bus structure is implemented with neurons clustered by local busses, grouped in global busses and finally in the WSI main bus.

HITACHI's **Back-Propagation Network** The described system has later been reworked and a new version supporting learning produced ([YAS91]). The approach used to perform the error back-propagation (multiplication by the transpose of the weight matrix) consists in duplicating the weight matrix in a second network dedicated to the backward processing. This is conceptually similar to the double storage implemented in [McC91] but here the whole neurons are replicated in hardware. The weight update is performed on both $\mathbf{W}$ and $\mathbf{W}^{\mathrm{T}}$ at the same time, so to guarantee that the matrixes are kept identical.

The internal structure of the processing elements is slightly modified to accomodate a few more primitives and the weight update process. A few additional registers are added: The weights are now stored on 16 bits and a matrix of $\Delta\mathbf{W}$ is stored on 8 bits to implement a momentum term. The reduced weight table described for the feedforward-only network is implemented also in the back-propagation processing element. In the present case, apart from the algorithmic impact of the feedforward solution, another key problem is not described in the paper: How is it performed the update of weights not present in the table? It is very possible that this simplified update scheme may strongly worsen the convergence of the back-propagation algorithm, which is often reported as being rather sensitive to the precision. No final statement on the subject can be made, the paper missing any information on the subject. Once again, the biggest interest of the machine is the hardware implementation on a 5" wafer, the basic characteristics being similar to those of the feedforward-only design.

2.1.3 PHILIPS' Lneuro 1.0

The Lneuro 1.0 architecture has been described in [THE90] and in [MAU92]. Both descriptions are rather general and insufficiently detailed for an accurate analysis. The integrated circuit has been also used by Edoardo Franzi at LAMI-EPFL in a compact back-propagation co-processor board for a 68030-based machine. The document describing this project ([FRA92]) is an excellent source of information on the chip and will be used in the sequel as a primary reference.

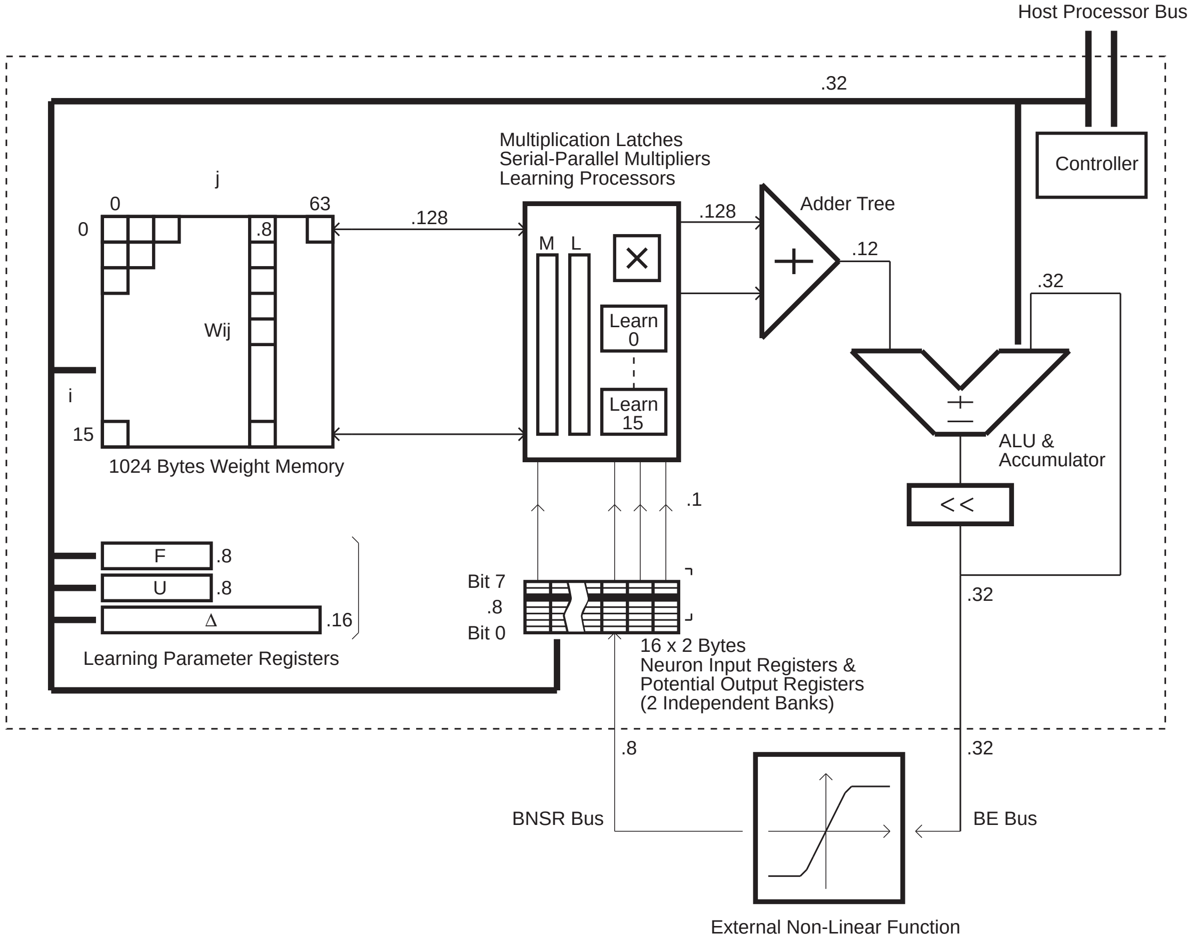

Figure 5. LNEURO. Processing Element Structure.

Figure 5 (courtesy of [FRA92]) shows the structure of a single Lneuro Processing Element. As it is also the case with the CNAPS architecture (see §2.1.1), it is essentially a *stored weight* architecture, in the sense that the weights of each neuron are resident inside the machine. In the forward phase, a single serial-parallel multiplier performs the products and sums required in the matrix-vector product $p_j = \sum_{i=0}^{n-1} \mathbf{W}_{j,i} \cdot x_i$. Each neuron (corresponding to row j of $\mathbf{W}$, whose transpose is represented in figure 5) is processed sequentially, as figure 6 shows. This is also done with the intent of producing a single potential output at a time, allowing to time-share the external hardware dedicated to the non-linear function $\sigma(\bullet)$ (typically a look-up table).

The 1 Kbytes of on-chip RAM are arranged as the 8-bit weights for 64 neurons ($0 \le i < 64$) having 16 inputs each ($n = 16$). A variable-precision technique allows to rearrange the weights for 256 4-bit neurons. During the learning process, 16-bit weights are used (two 8-bit words are coupled), reducing the maximum number of neurons to 32. A distinctive element of the architecture is its easy interfacing with a μP via the host standard bus. Lneuro executes microcoded instructions received through the bus and mapped on the host address space. The circuit shares the processor clock at 25 MHz and has an additional 50 MHz synchronous clock. Typical instructions reading or writing internal Lneuro registers or transferring data inside the

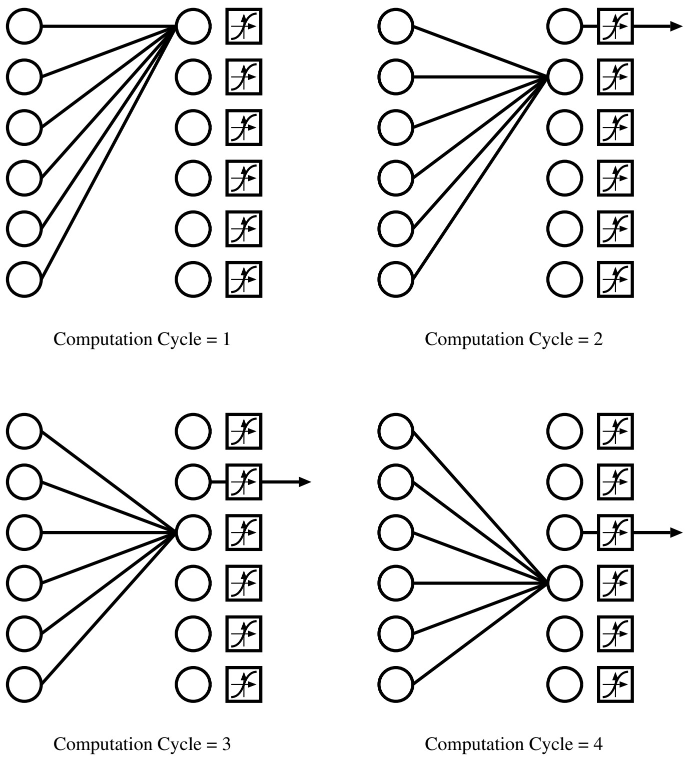

Figure 6. Lneuro. Operation Parallelization.

circuit take a single processor cycle. Complex operations as the computation of the neuron potential require some instructions to prepare the data. When the computation instruction has been issued by the host processor, the details of the operation are microcoded in the Lneuro and the host is kept free for other activities during some cycles (approx. 625 ns, equivalent to some 15 cycles). One interesting aspect in the forward phase is the duplication of the neuron input register file. The two parts are completely independent and only one is used for the potential evaluation. This implies that the second can be accessed during the computation to download the output of the external non-linear function. In other words, this allows a simple pipelining between the potential computation and the look-up table search (as shown in figure 6). The throughput of the forward phase is thus $\theta = 1/\max(\tau_p, \tau_\sigma)$ where τ_p is the time taken by the Lneuro to compute the potential and τ_σ is the time taken by the external hardware to compute the value of the non-linear function. In the implementation described in [Fra92] the function evaluation takes something less than 500 ns, thus allowing one neuron evaluation every 625 ns.

The weights stored on-chip and the single input path to the weight memory (i.e., the host processor bus) create the same problems for virtual networks that have been discussed for the Adaptive Solutions chip (see §2.1.1). Anyway the initial value of the accumulator can be initialized to a downloaded value (through the host bus), allowing the computation of the potential for neurons with more than 16 inputs. This can be done either by cascading two computations on the same chip, or by pipelining the output of a circuit to the accumulator of the other.

The learning functions have some special hardware available. Essentially, the operation which is allowed inside the chip is $\mathbf{W}_{ij}^{\mathrm{NEW}} = \mathbf{W}_{ij}^{\mathrm{OLD}} + \Delta \cdot x_j$ for $0 \leq j < 16$. It is performed in parallel (in sixteen identical circuits) over the sixteen values of j, that is over a single row of weights or a single neuron. Δ is a single value for all these weights and can be changed for each row, as required by the back-propagation algorithm. Other registers (namely F and U) are used for this calculation and control the details of the data handling. Even though they allow a large freedom in this process, they seems to be of practical utility only to force a correct 2's complement handling and to impose a coefficient of 0, 1 or -1 to the $\Delta\mathbf{W}$ value.

The implementation of the back-propagation algorithm would also require, to compute δ, the term to term multiplication of the output error vector by the derivative of the non-linear function (see equations (3) and (4)). In principle, this could be done by the sixteen multipliers performing the potential calculation. Unluckily, the implementation of the multipliers makes them inseparable from the following accumulator stage, which is not required in this case. Thus, this operation has to be performed outside the chip, hindering the whole back-propagation performance. As it is often the case, there is a problem in back-propagating the error as this operation requires the use of $\mathbf{W}^{\mathrm{T}}$ instead of $\mathbf{W}$. In [Fra92], a solution conceptually similar to those described in [Yas91] and [McC91] has been taken: Two chips are used, one containing the weight matrix and the other containing its transpose. The relevant chip is used depending on the operation required and both matrixes are updated at the same time (and by the same amount). It is an effective solution (especially for small networks) but, as in the Hitachi design, it has the additional disadvantage of not only doubling the required memory but also of requiring the double of arithmetic units, one being always inactive when the other is computing.

Comparing the Lneuro with the CNAPS architecture, it seems that the former design shares some of the problems of the latter one without providing the interesting CNAPS generality. In fact, the advantages of the easy interfacing via a single input/output μP bus are responsible for the communication difficulties when dealing with virtual large networks not self-contained in one chip. Moreover, these advantages are outweighted by the rather rigid internal structure which prevents the reuse of the available hardware for other required operations (e.g., the product of the error on one neuron by the derivative of the non-linear function of the potential of the same neuron).

That said, it should be noted that the reduced complexity of the chip makes it useful for specific on-field feed-forward or back-propagation applications which are sufficiently small to be resident in the on-chip memory.

2.2 Ring Architectures

Many systems have been developed around a ring-shaped communication network and present a data circulation somewhat derived from the systolic ring-multiplier. Among them are:

- Lawrence Livermore National Laboratory's **SPRINT**. Described in [DeG89], it is a 64-element multiprocessor machine composed of INMOS Transputers. SPRINT is rather a general machine where the ring architecture has been mapped for the implementation of the back-propagation algorithm.

- University of Nottingham's **Toroidal Neural Processor**. The Toroidal Neural Processor ([Jon91a], [Jon91b] and [Hun91]), or TNP, is an SIMD array of processors, where each neuron is associated to an application-specific processor. All the processors (belonging to different layers of the networks) are ordered in a single ring structure and have each a memory-ring to store the corresponding weights (hence creating a toroidal structure).

- HNC's **SIMD Neurocomputer Array Processor**. HNC developed a custom floating point processing element (SNAP) dedicated to linear arrays ([Mea93]). The IC is used in a commercial board containing 16 PEs. It is essentially a ring-based machine but has some original characteristics such as a global bus to allow direct PE to PE interconnection. This feature will be discussed later in this section.

- **S. Y. Kung's and J. A. Vlontzos' Ring Array**. Two papers ([Kun90] and [Vlo91]) present a ring-array of processing elements built around a commercial Weitek floating-point unit.

- ICSI's **Ring Array Processor**. Described in [Mor92], it is composed of cards with four TMS320C30 processors each. Systems with up to 40 nodes are reported. The communication between nodes is asynchronous.

- ETHZ's **Music System**. [Mul92] describes a general-purpose multiprocessor system using Motorola 96002 Digital Signal Processors. The communication among the processors is done via a pipelined ring bus. The paper reports the use of the system for NN computation. The algorithm implementation details are reported and the possible uses of the connection network in the different implementations is discussed.

At an architectural level and considering the data flows in the systems, the differences are relatively minimal. The SPRINT implementation will be discussed in the sequel to show the flow of data on the ring during the various NN computations. It is one of the earliest and more accurate references for the purpose. The TNP and the SNAP will be shown afterwards because they display interesting features. In contrast with the SPRINT, where the ring is mapped on commercial processors, both the TNP and the SNAP are custom processing elements.

The Siemens' SYNAPSE-X system is not usually presented as a 1D structure but as a 2D one. In fact, it is built on systolic 4×4 matrix processors connected on a ring. The ring structure can be replicated to improve throughput obtaining a *cylindric* array which is very different from other 2D structures assigning one processor per synapse. For this reason, the Siemens architecture will be addressed in this section.

2.2.1 Lawrence Livermore National Laboratory's SPRINT

As mentioned, the SPRINT design will not be covered for the hardware implementation itself but to show how the operations needed for NN can be mapped on a ring structure ([DeG89]). First-order gradient descent methods require the following operations:

1. matrix-vector multiplication;
2. transpose matrix-vector multiplication;
3. outer product (vector-transpose vector multiplication);
4. term to term vector-vector multiplication;
5. scalar multiplication;
6. function application to all elements of a vector;
7. matrix addition; and
8. vector addition.

If there are as many processors as rows in the matrix and vectors referred to in the list, the operations # 4 to # 8 can be performed in parallel by each processor as it contains the necessary data. Note that the number of rows in the weight matrix corresponds to the neuron count, that is a number of processor equal to the number of neurons is supposed. The ring structure is used to implement the first three operation, as shown in figure 7. Above the processing elements are represented the local data used for the computation. To perform the matrix-vector multiplication, the vector elements are circulated into the ring. Each processor multiply the received element by the appropriate weight and accumulates the result until all the elements have been used. This mode essentially corresponds to the broadcast bus mode but with the input bus replaced by the ring. The only practical difference is that the processors are active on diagonal elements of a matrix instead of being active on a row. The transpose matrix-vector multiplication is performed similarly but now the vector elements are kept local and the accumulator result is kept circulating. Finally, the outer product is performed by rotating in the ring one of the vector and keeping the other resident into the processors. Term-to-term multiplication is performed every cycle, obtaining the elements of one row in each processor.

As mentioned this flow of data is rather typical in all the ring implementation and, most notably, is the key of the second hierarchical level of the Siemens architecture.

2.2.2 University of Nottingham's Toroidal Neural Processor

Three papers have been reviewed on this architecture: [Jon91a], [Jon91b] and [Hun91]. While the last one is mainly concerned with the problems posed by the array controller, the first two describe the processor. There seems to be a strong regularization (orthogonalization) of the processor in the second paper compared to the first. Purpose dedicated registers become perfectly interchangeable in their functionality, and all the structure more versatile. The processor structure is shown in figure 8. It contains eighteen 16-bit general-purpose registers and four dedicated registers. A couple of remarks on the processor and its instruction set can be made:

- The processor architecture is extremely simple and regular, probably with advantages both for the processor implementation and the processor automatic sequencing (high programmability).
- Each processor contain a comparator unit producing a 0 or a 1 output. The result can be put in a register and reused in the next cycle to perform conditional statements. These are adapted to the pipelined structure of the ring as the instruction sequence remains constant. At the same time it is expected that they give an further interesting degree of programmability to the system.

A few elements are specifically included in the structure to support the distinctive characteristic of the TNP, that is the fact of modelling a multi-layer network with one processor per neuron allocated in the same ring, independently of the layer. The processors contain a `Tag` register which identifies the layer to which the neuron belong. Each processor executes to the instruction issued only if the register contents matches the corresponding field in the instruction code.

A very interesting aspect of the TNP is that the University of Nottingham and the Technical University of Denmark are jointly developing a delay-insensitive version of the TNP ([Nie91]). The asynchronous design has two important advantages: First, the design operates at the real circuit *average* speed instead of the *worst-case* critical-path propagation delay used to determine the clock period. But a second advantage may play, for very large networks, a more important role: The design has no global clock, thus eliminating all the

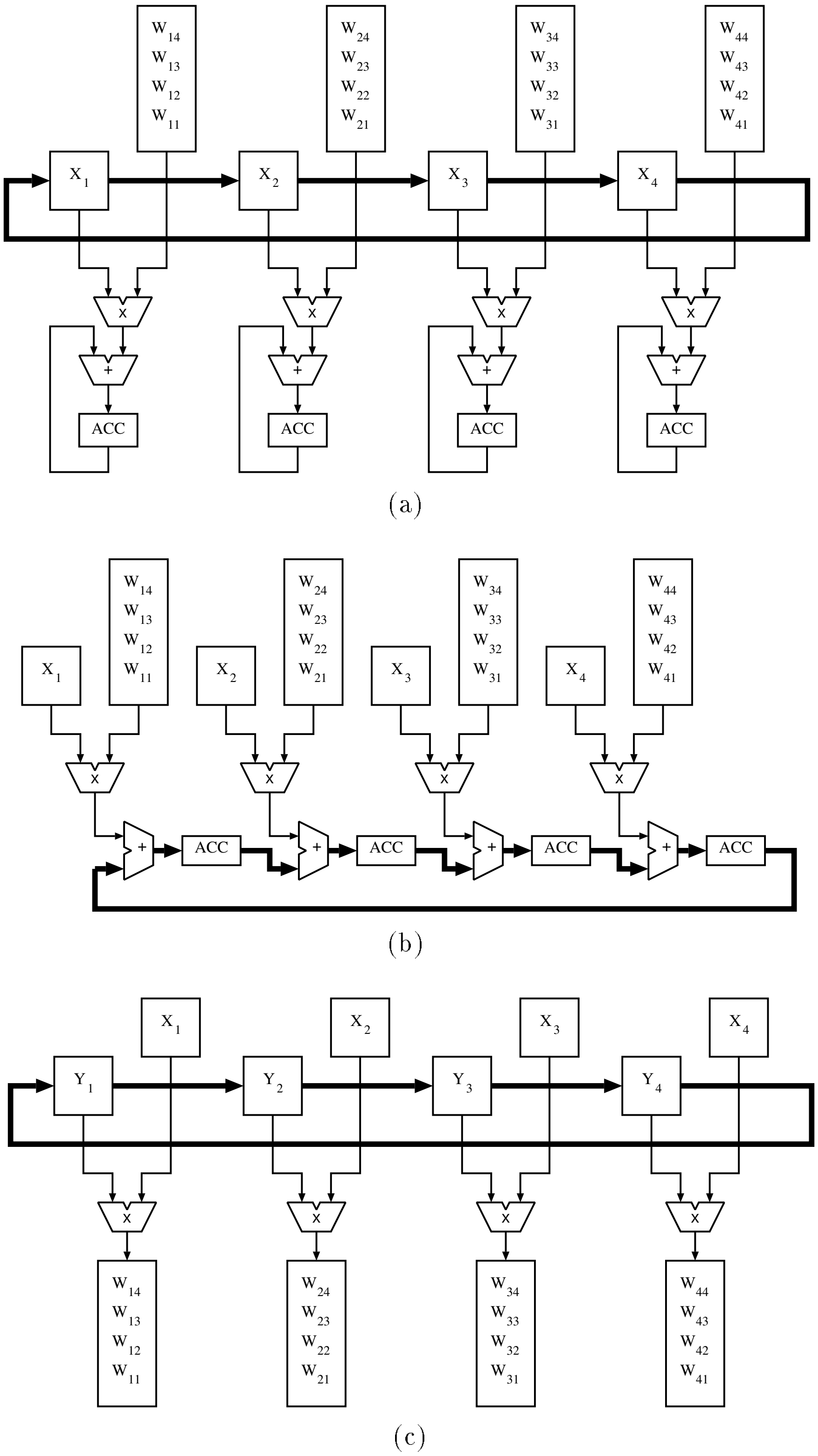

Figure 7. SPRINT. An example of a ring array composed by four processing elements. Detail of the three operation modes (snapshot before the beginning of the computation): (a) matrix-vector multiplication; (b) transpose matrix-vector multiplication; (c) outer product (vector-transposed vector multiplication).

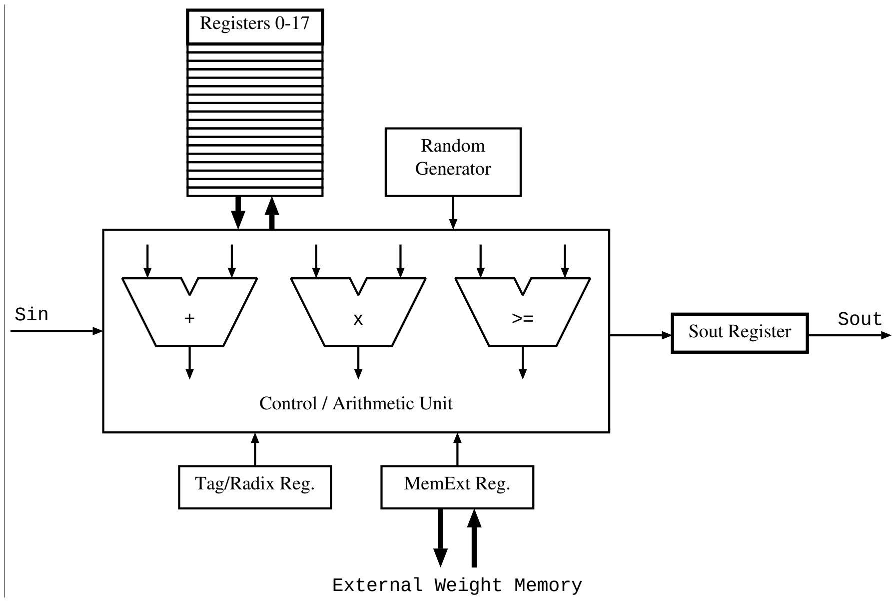

Figure 8. TNP. Processing Element.

connected problems as the skew. Addressing this kind of improvement—which interestingly does not affect the general data architecture—is beyond the scope of this document.

2.2.3 HNC's SNAP

The best source of information on the SNAP architecture is [Mea93], which extends the contents of the original presentation of the architecture ([Mea91]). Some technical details on the processor IC may be found in [HNC93].

The processing element has a rather simple structure but is conceived to maximize generality. As figure 9 shows, the core of the PE is composed of a 32-bit floating-point multiplier and a 32-bit floating-point and integer ALU. Around that, there are seven data registers, three instruction registers, and one status register. Many 32-bit interconnections allow a versatile data routing among the different elements. The communications with the memory and the other PEs are performed through four channels: (a) a 32-bit bidirectional data path to and from the PE local memory; (b) a 32-bit bidirectional data path to and from a memory bank global to the whole machine; (c) two 32-bit monodirectional data path connecting each PE with the nearest neighbours (monodirectional ring[4]). The resulting structure of the system is shown in figure 10.

Two elements make the HNC architecture interesting for the purpose of this analysis:

- The explicit interest of HNC in producing a very general machine makes the processing element as general purpose and as flexible as possible. This implies that the instructions to the SIMD array are coded on *Very Long Instruction Words (VLIWs)* of 96 bits. An IDT microsequencer and two IDT 16-bit microprocessor slices are used to sequence the system and to produce the local bus and global bus addresses. The instruction set of the SNAP

[4]Note that some figures of [Mea93] (e.g., figures 1 and 2) show a bidirectional ring. This is in contrast with the description of the PE and with the content of [Mea93, Figure 3].

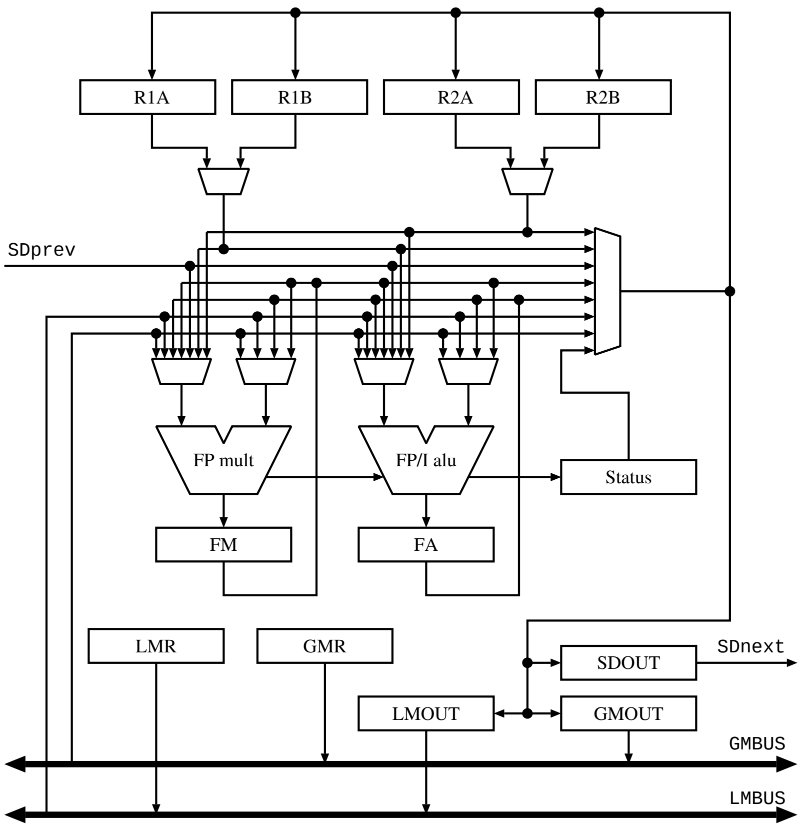

Figure 9. SNAP. Processing Element. All data paths are 32-bit wide.

has some characteristic features such as bits of the code word devoted to the *Conditional Execution Control (CEC)* and the *Halt Mask Control (HMC)*. In the PE, some flags of a status register can be set depending of the result of arithmetic operations. The bits in the CEC allow instructions to be executed only if a flag has a given value, thus providing a sort of "IF condition THEN instruction" structure. Similarly, the bits in the HMC allow to halt the processor until the mask is explicitly cleared. This results in the possibility of executing program flows of the type "IF condition THEN block of code ELSE block of code". These features establish the SNAP as one of the most programmable ring-based systems.

- An element that differentiates the SNAP from all the other ring architectures considered here is the presence of the global interconnection bus. This is not simply an interface to a global memory but is designed to allow a direct PE to PE interconnection. [Mea93] provides a numerical example of the usefulness of this channel. The problem is here generalized to the need of adding N elements, each computed and stored in one of the N processors. Typically, $N-1$ cycles are required in a ring structure: All the processors but one simply rotate the elements on the ring, only one processor being active as an accumulator; this implies a bad parallelization of the operation and a long latency. With the global communication available, it becomes more efficient to have some partial accumulators equally spaced on the ring (e.g., n accumulators require $\lceil N/n \rceil - 1$ cycles) and

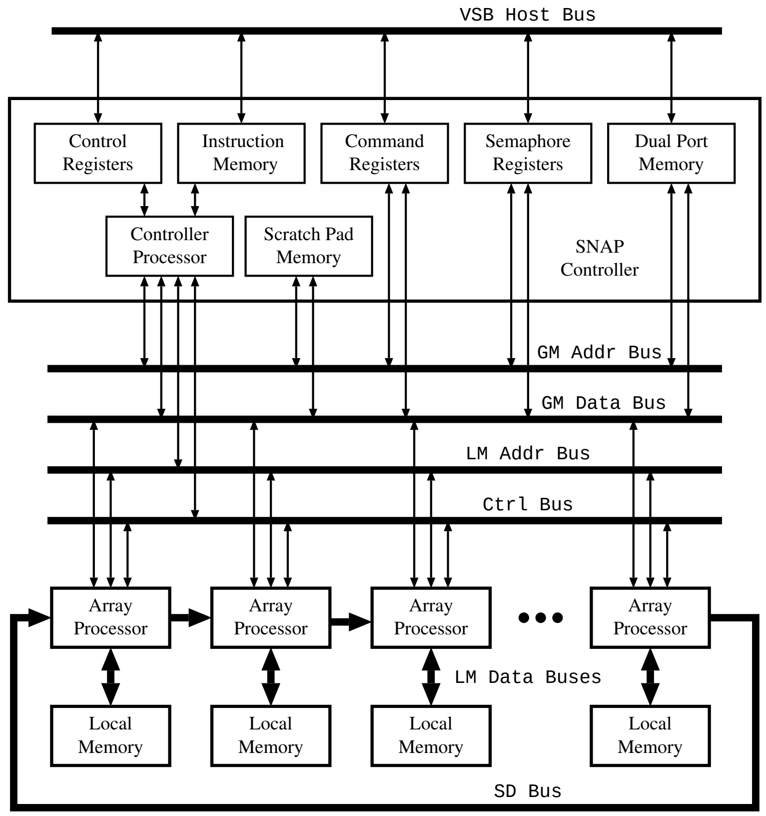

Figure 10. SNAP. System Architecture.

then to add together the n partial sums through point to point communication (in further $n-1$ cycles). It is easily verified that appropriate values for n minimize the number of cycles required for the operation. In the example shown in [MEA93], 31 cycles to sum 32 elements on 32 processors are reduced to 10 total cycles to partially accumulate over 4 PEs (7 cycles for the partial accumulation and 3 cycles for the partial sum broadcast). As it will be shown in section 3, the rigidity of the standard ring structure is, in practice, not much inferior to that of 2D architectures; features like the described global bus may substantially improve the system performance in critical cases.

2.2.4 SIEMENS' SYNAPSE-X

The SIEMENS' architecture and the systolic concepts on which it relies have been described in several papers ([RAM90], [RAM91B], [BEI91B], [RAM91A], [RAM91D], [BEI91A]). The paper [RAM91B] is probably the most complete and will be used as a reference in most of the cases.

As it is the case with many NN hardware implementations, the SIEMENS approach descends from an investigation of several common neural algorithms ([RAM89]) and from a mathematical unification of the different learning algorithms ([RAM91C]). This results in the decision to implement some computation intensive operations in specialized hardware. The list of these operations contains some inconsistencies between the quoted papers and the notation is often not made clear enough. Table 1 appears to be the most reliable list of the operations targeted

Operation	Comments
$S_{i,j} = \alpha_i \sum_k A_{i,k} \cdot B_{k,j}$ $S_{i,j} = \alpha_j \sum_k A_{i,k} \cdot B_{k,j}$	Matrix-matrix product with a different coefficient on each row or column
$S_{i,j} = (\sum_k A_{i,k} \cdot B_{k,j})^2$	Matrix-matrix product with each element squared
$S_{i,j} = \alpha_i (A_{i,j} \pm B_{i,j})$ $S_{i,j} = \alpha_j (A_{i,j} \pm B_{i,j})$	Matrix-matrix sum/subtraction with a different coefficient on each row or column
$S_{i,j} = (A_{i,j} \pm B_{i,j})^2$	Matrix-matrix sum/subtraction with each element squared
$S_{i,j} = \lvert A_{i,j} \pm B_{i,j} \rvert$	Matrix-matrix sum/subtraction absolute value of each element
$S_{i,j} = A_{j,i}$ $S_{i,j} = B_{j,i}$	Matrix transposition
$r_i = \sum_j S_{i,j}$ $r_j = \sum_i S_{i,j}$	Sum of the components of each row or column
$r_i = min_j S_{i,j}$ $r_j = min_i S_{i,j}$	Search for the minimum components of each row or column
$r_i = max_j S_{i,j}$ $r_j = max_i S_{i,j}$	Search for the maximum components of each row or column

Table 1. SYNAPSE-X. Operations Targeted by the Architecture.

by the authors (adapted from [Bei91b]).

One distinctive element of the architecture is the hierarchical approach taken in partitioning the problem. The architecture is better explained by considering the three architectural levels one at time:

1. Elementary Chains;
2. MA16 chip-level and ring arrays of MA16; and
3. multiple ring arrays of MA16.

Elementary Chains. At a first level, an *Elementary Chain* produces some of the results shown in Table 1 on 4×4 matrixes.

An Elementary Chain is represented in figure 11. It contains a linear systolic array composed of four multipliers and four adders to perform both matrix-matrix multiplication and addition on 4×4 inputs. The array is fed by two set of 16 registers holding a local copy of the 4×4 matrixes to process (indicated as $\mathbf{A}$ and $\mathbf{B}$ in the sequel). A shadow copy of these registers can be fed (respectively through `In1` and `InOut2`) with the next input matrixes while the previous operation is taking place. The multiplication array in ALU1 normally performs the operation $\mathbf{S} = \mathbf{B} \times \mathbf{A}$ or, by modification of the $\mathbf{B}$ element scheduling, $\mathbf{S} = \mathbf{B}^{\mathrm{T}} \times \mathbf{A}$. Some switches allow it to perform the $\mathbf{S} = \mathbf{I_4} \times \mathbf{A} + \sigma_1 \cdot \mathbf{B}$ instead, where $\mathbf{I_4}$ is the identity matrix of size four and σ_1 is either 1 or -1. These operations are performed with a throughput of one operation every 16 cycles, thus with the same rate needed to feed the input registers. The 16 outputs of this stage (i.e., the elements of the resulting 4×4 matrix) are either multiplied by four factors (a different one each row or column) or squared (by multiplying them by themselves) using the multiplier in ALU2. The final stage ALU3 accumulates the result in a third 16 register set: It is configured to perform the operation $\mathbf{C} = \mathbf{S} + \sigma_2 \cdot \mathbf{C}$, where σ_2 is either 1 or -1. The accumulation can be either performed locally on the matrix contained in the register, or performed on the data

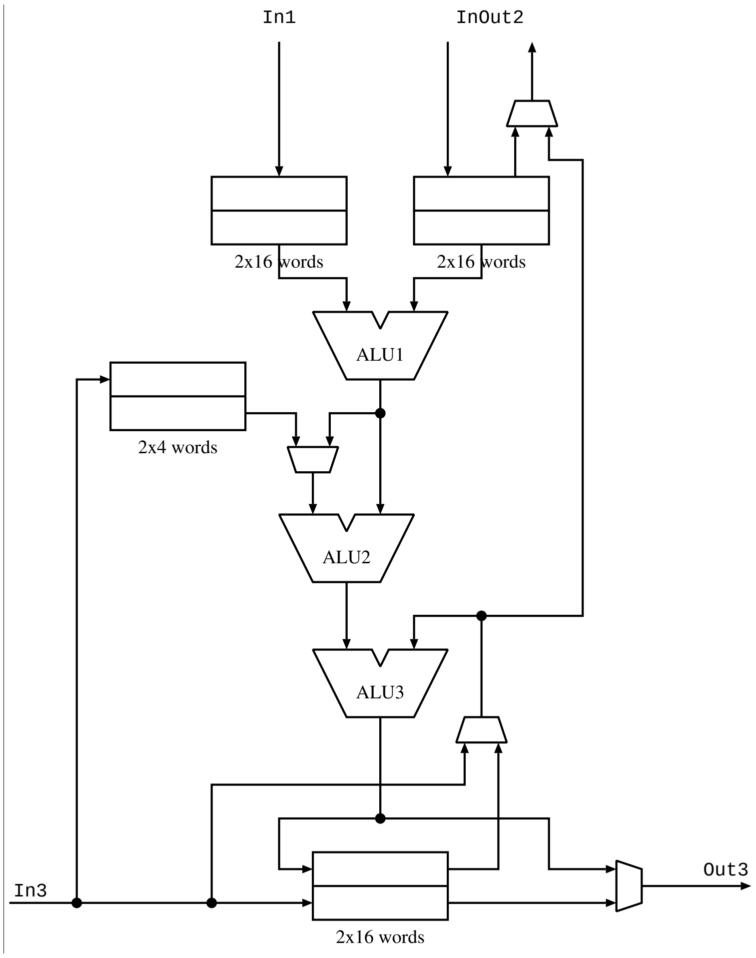

Figure 11. SYNAPSE-X. Elementary Chain.

coming from `In3` and the result forwarded to `Out3`. Again, some switches allow the modification of the functionality of this stage to perform the minimum/maximum search. From [Bei91b], it seems that the operation consists in finding a matrix whose elements are either the minimum or the maximum between the corresponding elements of the two input matrix (e.g., in symbols, $C_{i,j} = max(S_{i,j}, C_{i,j})$). In fact, the utility of this operation is not manifest: the operation needed for maximum or minimum search in competitive learning usually requires the identification of the winning unit rather than just the choice of the extremum.

MA16 Chip and Ring Array of MA16. Each chip contains four Elementary Chains connected as shown in figure 12. This connection style is intended to be replicated at system level (see figure 13). The `INOUT2/x` signals are interfaced directly with DRAMs, while `IN1`s are connected to taps along a delay line and `IN3`s and `OUT`s are connected in a loop including an extended accumulator (`FIFO`) and an arithmetic unit (`ALU`).

The obtained structure is a ring array whose elementary processors are not scalar processor but rather 4×4 matrix arithmetic units. As mentioned in §2.2.1 about SPRINT, during matrix-vector multiplication (forward phase), the processor multiplication results are locally

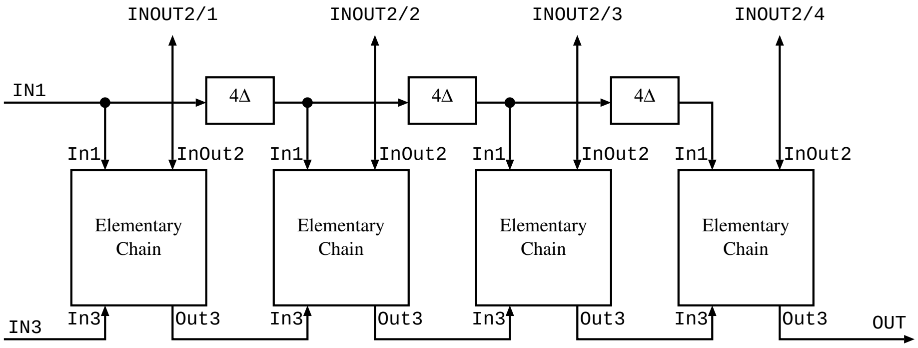

Figure 12. SYNAPSE-X. MA16 Chip Structure.

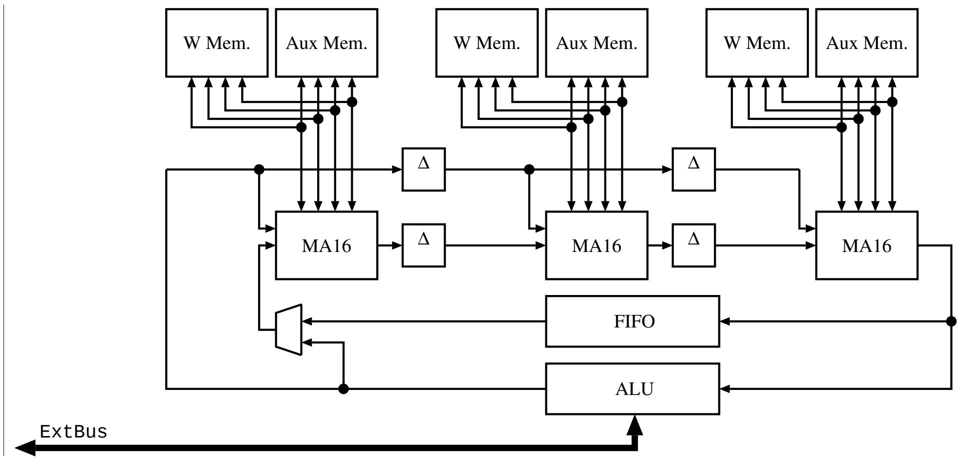

Figure 13. SYNAPSE-X. Ring Array of MA16 Chips.

accumulated. During the back-propagation phase (i.e., using the transpose matrix) the accumulation is done on the elements circulating in the ring and the `FIFO` is present to increase the ring length and thus allow for non-square weight matrix aspect ratio. In fact, figure 7(b) in §2.2.1 is rather tricky in that it presents a square matrix and avoids the problem. In general (considering scalar processing elements), for an $m \times n$ matrix, only m processors are needed (as many as the neurons) if a row is stored in each processor. Anyway, the input vector is composed of n elements and the transpose multiplication results in a vector containing n elements; thus, if $n > m$, a longer ring is required.

Multiple Ring Arrays of MA16. To improve the performance of the system, three possibilities can be taken into account:

- More MA16 can be inserted into the ring. This increases the computation power and the maximum size of the networks that can be implemented[5].

[5]In fact, without specific scheduling techniques not mentioned in the original papers, the increased computational power can be used only for larger networks.

- More weight memory can be added to each MA16. This does not modify the system throughput but has again a positive impact on the maximum size of the implemented networks.
- The ring of MA16 processors can be replicated and the weight memory shared by corresponding chips of different rings.

It is important to note that the first improvement (almost quoted from [RAM91B]) implies that no virtual matrix support is (or was at the time of writing) foreseen for the architecture. Thus, as in a ring each processor represents a neuron (when local accumulation is chosen in the forward phase), each MA16 in the ring implements 16 neurons. The maximum neuron count in the network is then limited to $16N_{\mathrm{MA16}}$, where N_{MA16} is the number of MA16 in the ring. The maximum number of inputs is instead determined by two separate elements: (a) availability of weight memory, and (b) length of the ring (including the FIFO). The second improvement in the above list implies a possible increase in the maximum input count. This is given by $\min(16N + 4N_{\mathrm{FIFO}}, 4N_W)$, where N_{MA16} is again the number of MA16 chips, N_{FIFO} is the depth of the FIFO in 4×4 submatrixes and N_W is the number of weight 4×4 submatrixes per Elementary Chain which are available in the weight memory. The last improvement (shown in figure 14) has no impact on the network size but simply multiplies by a factor N the forward throughput, where N is the number of ring replicas. The speed-up obtained by replication of the network is probably well below linearity (no data are given in the papers) for the back-propagation phase due to the need to update the single ring of weights with the contributions computed by each ring. This is done via a systolic propagation of the $\Delta\mathbf{W}$ submatrixes to the topmost ring.

Three final remarks:

- The SIEMENS architecture is very interesting for the versatility of the processing element and for the highly adapted weight memory interface[6].
- The described structure seems to be well modular (especially when multiple rings have to be connected in parallel) and new operating modes should be implementable by a simple processing element or ALU upgrade, the system architecture seeming sufficiently general to allow for future algorithmic expansions.
- The rationale for the first two levels of hierarchy (that is the decomposition in 4×4 submatrixes) is not directly discussed in the original papers ([RAM91B]). The point will be addressed in the analysis section.

2.3 2D Systolic Architectures

While monodimensional structures tend to allocate one processor to each neuron, typical 2D structures assign one processor per synapse. This has the immediate consequence of allowing—at least in principle—to exploit the full parallelism intrinsic to NN algorithms. Two systems will be considered:

[6] One important implementation feature of the SYNAPSE-X architecture has not been discussed in this document as it goes beyond the architectural scope of this text. Nevertheless, it should be mentioned the extreme care posed by the MA16 designers to provide the chip with an highly adapted interface to DRAM chips, that is with the cheapest and more dense memory chips on the market. It has been said that this feature is beyond the scope of this document as—at the abstraction level that will be used for the analysis—the ability to feed a communication channel with the necessary throughput (never above one word per cycle) is considered a prerequisite of the architecture and is not addressed in detail.

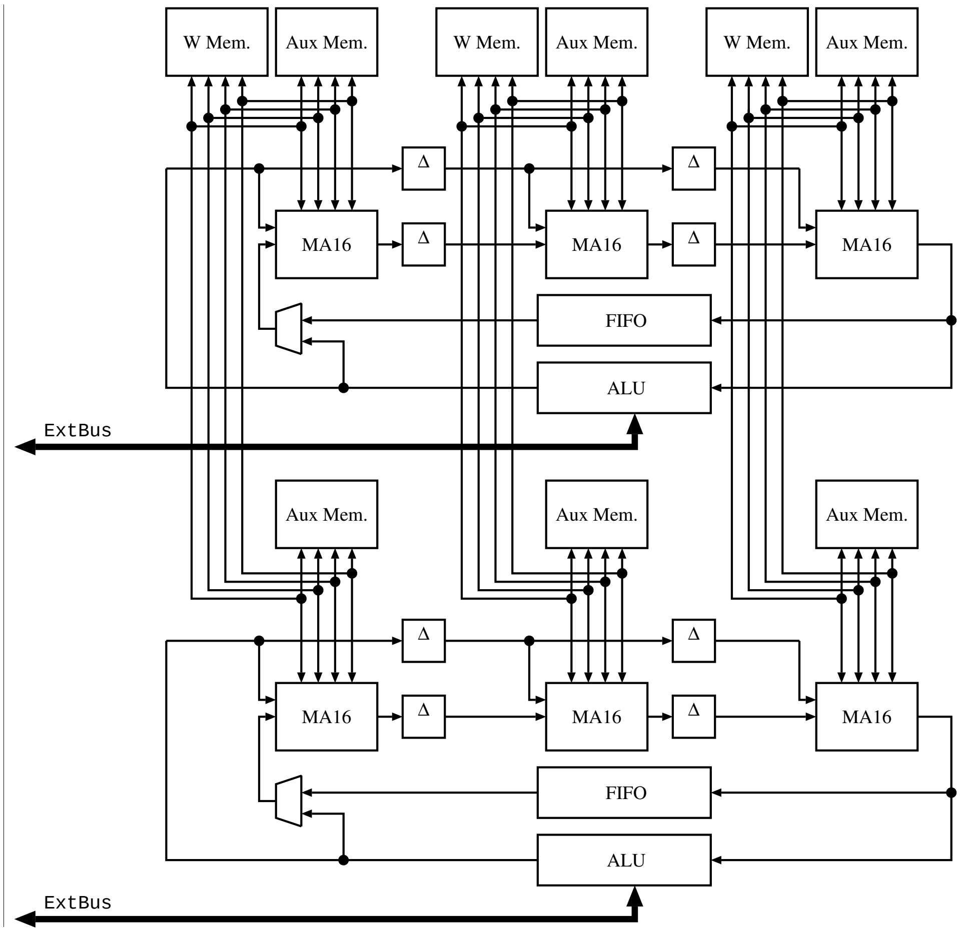

Figure 14. SYNAPSE-X. Detail of a Multiple Ring Array of MA16 Chips.

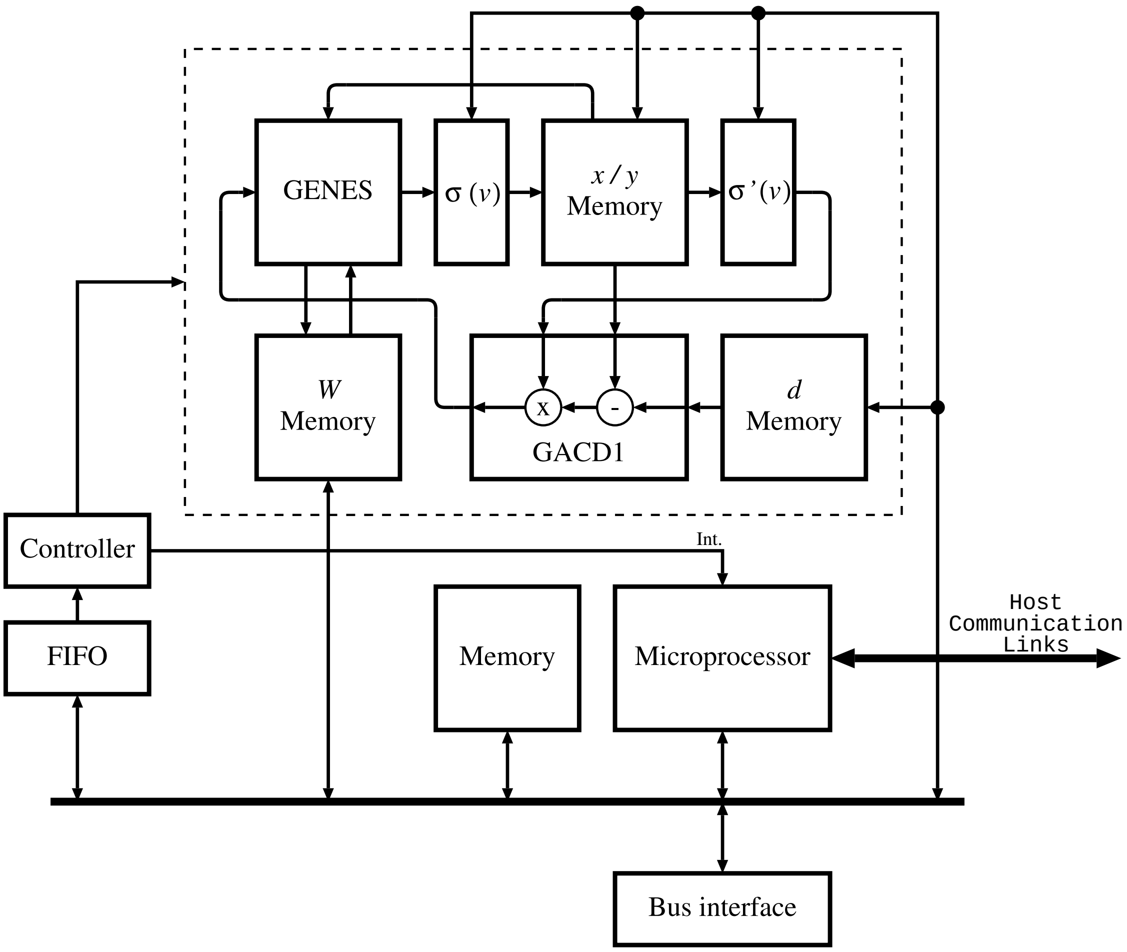

Figure 15. GENES IV. System Structure.

- The first one, originating at the Laboratoire de Microinformatique (LAMI) at the École Polytechnique Fédérale de Lausanne, is a multi-model square structure of serial-communication processing elements.
- The second, developed at the Korea Advanced Institute of Science and Technology, is a 2D systolic array completely dedicated to the back-propagation algorithm.

2.3.1 LAMI's MANTRA I and GENES IV

The MANTRA I system (currently under test at the LAMI) has been described in [VIR92]. The simplified structure of the machine is shown in figure 15. The computation heart of the machine is the square array of GENES IV systolic processors and the linear array of GACD1 auxiliary processors ([IEN92]). The internal structure of a GENES IV processor can be seen in figure 16. The implemented primitives allow to perform systolically the following operations:

- **Forward Phase**

$$p_i = \sum_j W_{i,j} \cdot x_j + y_i$$

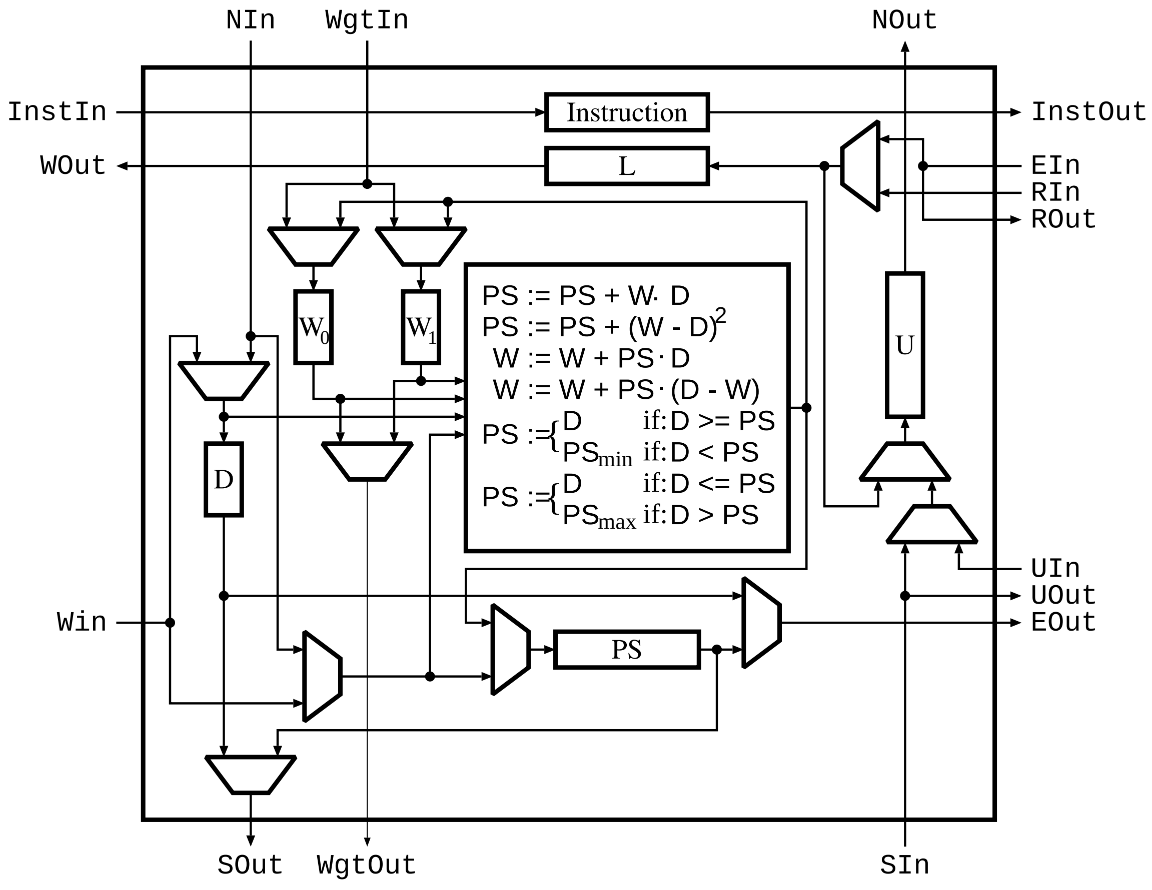

Figure 16. GENES IV. Processing Element.

$$p_i = \sum_j (W_{i,j} - x_j)^2 + y_i$$

- **Maximum and Minimum Search**

$$p_i = \begin{cases} x_i & \text{if } \max_j(x_j) = x_i \\ -\infty & \text{otherwise} \end{cases}$$

$$p_i = \begin{cases} x_i & \text{if } \min_j(x_j) = x_i \\ \infty & \text{otherwise} \end{cases}$$

- **Update Phase**

$$\begin{aligned} W_{i,j}^{\mathrm{NEW}} &= W_{i,j}^{\mathrm{OLD}} + y_i \cdot x_j \\ W_{i,j}^{\mathrm{NEW}} &= W_{i,j}^{\mathrm{OLD}} + y_i \cdot (x_j - W_{i,j}^{\mathrm{OLD}}) \end{aligned}$$

The forward operations can be performed also on the transpose weight matrix ($\mathbf{W}^{\mathrm{T}}$) by exchanging the horizontal and vertical data flows in the array. The GACD1 monodimensional array performs other operations required for the back-propagation, such as the difference between the desired and the real output. The latency of the 2D array is $2N$ cycles, to which the latency of the GACD array and of the non-linear function table should be added. Given a throughput of one vector per cycle, this asks for the processing of groups of something more than $2N$ vectors at the same time. These groups will be called *epochs* in the sequel. For large N, the need to have long epochs may be annoying, especially in some applications. An important feature of the system is the duplication of the weight register in the processors, which allows for a weight matrix swap in the background, while a computation is taking place. This feature is key to an efficient support for virtual networks.

A limitation of this architecture is—when dealing with some non-recurrent algorithms, such as the back-propagation rule—the fact of relying heavily on a single square matrix of PE. The square $N \times N$ matrix scales well to relatively large values of N. Unfortunately, typical applications do not scale with the same 1:1 ratio: Consider the significance of the size of the two sides of a GENES IV when mapping a Kohonen network on it. The horizontal side (normal direction of PS propagation) represents the number of inputs while the vertical side represents the number of neurons (one neuron per matrix row). Large Kohonen networks may well have neuron counts up to 10000 (100×100 arrays) but seldom the input space has more than, say, 10 dimensions. Scaling up today's GENES IV arrays to accommodate 25×25 Kohonen nets with a 10-dimensional input space implies obtaining intrinsic PE utilisation efficiencies of $\frac{10}{25^2} \cdot 100 = 1.6\%$. This sort of measurement can be formalized by defining two quantities: the *Parallelism Exploitation Efficiency* or η_P and the *PE Utilisation Efficiency* or η_U. The former measures the ratio between the throughput of the machine and the optimal throughput (i.e., the throughput of the best possible machine, which—exploiting the whole parallelism of the problem—solves it once per cycle). The PE utilisation efficiency is the ratio between the number of simple operations in the problem, which is the minimum (optimal) number of operations to perform to solve the problem, and the number of operations actually performed by the system. In symbols, for GENES IV, we have:

$$\eta_P = \frac{\text{Real Throughput}}{\text{Optimal Throughput}} =$$

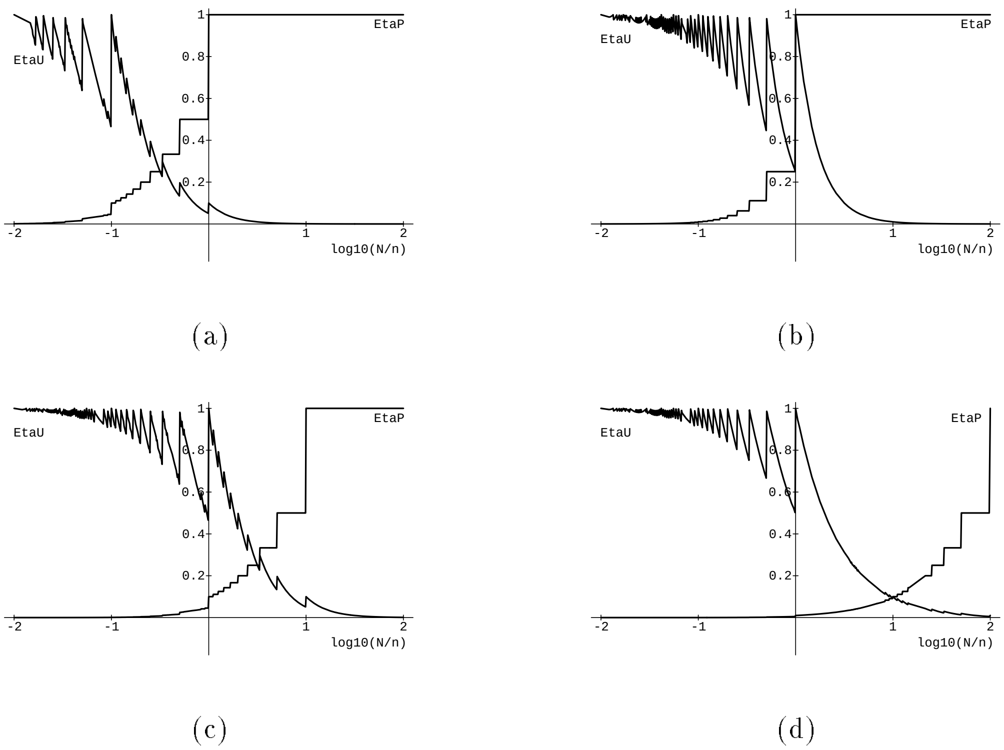

Figure 17. GENES IV. Plots of η_P and η_U in function of the order of magnitude of the ratio between the array size N and the number of columns in the matrix n. The four plots correspond to different matrix aspect ratios: (a) $\left(\frac{m}{n}\right) = .1$; (b) $\left(\frac{m}{n}\right) = 1$; (c) $\left(\frac{m}{n}\right) = 10$; (d) $\left(\frac{m}{n}\right) = 100$.

$$
\begin{aligned}
&= \frac{1}{\lceil n/N \rceil \cdot \lceil m/N \rceil} &&= \frac{1}{\left\lceil \left(\frac{m}{n}\right) \cdot \left(\frac{N}{n}\right)^{-1} \right\rceil \cdot \left\lceil \left(\frac{N}{n}\right)^{-1} \right\rceil}; \\
\eta_U &= \frac{\#\ \text{Elem. Ops.}}{\#\ \text{Elem. Ops. Performed}} &&= \\
&= \frac{n \cdot m}{\lceil n/N \rceil \cdot \lceil m/N \rceil \cdot N^2} &&= \frac{\left(\frac{m}{n}\right) \cdot \left(\frac{N}{n}\right)^{-2}}{\left\lceil \left(\frac{m}{n}\right) \cdot \left(\frac{N}{n}\right)^{-1} \right\rceil \cdot \left\lceil \left(\frac{N}{n}\right)^{-1} \right\rceil}
\end{aligned}
$$

where:

- $m \times n$ is the size of the weight matrix (typically m is the number of inputs to the network and n the number of neurons, or outputs);
- $N \times N$ is the size of the physical array of GENES IV processors; and consequently,
- $\left(\frac{m}{n}\right)$ is the aspect ratio of the weight matrix (rows over columns);
- $\left(\frac{N}{n}\right)$ is the ratio of the size of the GENES IV array and the weight matrix column count.

Figure 17 shows these two quantities plotted for four values of the weight matrix aspect ratio. It can be noted that the relationship between the two efficiencies depends only on the absolute value of the aspect-ratio order of magnitude ($|\log_{10}\left(\frac{m}{n}\right)|$), the sign accounting only for a translation. If one considers a given problem or class of problems (say, defined by a given $\left(\frac{m}{n}\right)$), η_P shows—from left to right—the efficiency of growing machines in addressing the problem. η_U shows how well the added hardware is exploited in larger machines: As the machine size grows compared to the problem, the added value is worse exploited. Interestingly, high values of $|\log_{10}\left(\frac{m}{n}\right)|$ correspond the wide gaps between the two optimal efficiencies, that is, the cost-performance trade-off becomes burdensome.

A typical performance index of parallel systems is the *speed-up* S defined as

$$\begin{aligned} S &= \frac{\text{Real Throughput}}{\text{Single PE Throughput}} = \\ &= \frac{n \cdot m}{\lceil n/N \rceil \cdot \lceil m/N \rceil}. \end{aligned}$$

Its ratio with the number of PEs is called *Parallelization Efficiency* η_Q and indicates the performance improvement obtained by adding more PEs weighted against their cost:

$$\begin{aligned} \eta_Q &= \frac{\text{Speedup}}{\text{\# of Processing Elements}} = \\ &= \frac{n \cdot m}{N^2 \cdot \lceil n/N \rceil \cdot \lceil m/N \rceil}. \end{aligned}$$

Due to the absence of communication overhead, this is equal to η_U.

This new interpretation of η_U allows to gain another view of the same problem. Figure 18 shows η_Q plotted with η_P on the same graphs. The first plot shows the situation in case of a square matrix ($m = 10$ and $n = 10$): the performance/cost indicator η_Q can be maximized (1) and at the same time the best performance reached (with 100 PEs). If the aspect ratio changes (see figure 18(c) for a ratio of 100), still all the parallelism intrinsic to the problem can be exploited (with one million of PEs) but at an extremely low point of the performance/cost graph.

The conclusion is that, while the GENES architecture is very well scalable in principle, for large real problems (where it is possible to have aspect ratioes of 1000 or more), it is not envisageable to scale it over some dimension which essentially corresponds to the average minimum dimension of the weight matrix: For a number of PEs larger than the square of this value, the performance/cost ratio is rapidly decreasing. Of course, all this applies to the current structure of the array which is necessarily square. Other techniques could be investigated to rely on a set of smaller arrays rather than on a single, very large array.

It is interesting to note the ability to rely efficiently on medium size square arrays may be winning also considering technologic constraints. A simple computation can provide an approximate 3-5 years ahead prediction of maximum arrays technically feasible in WSI. The current size of a PE (1.0 μm CMOS technology, VLSI Technology Inc. VSC370$^{\text{TM}}$ Standard Cells) is $2039\mu\text{m} \times 1809\mu\text{m}$, that is, approximately, four square millimeters. The following hypotheses can be enforced: It can be foreseen a 2-3 times complexity increase in the processor to improve its generality. A reduction in the feature size from 1.0 μm to 0.5 μm can be expected, while an improvement in silicon implementation efficiency may account for a factor 3-4 reduction in surface. Roughtly a PE could occupy a square millimeter. On a 50 mm $\times$ 50 mm surface, some 2,500 PE could be integrated. If an overhead factor of 3-4 is assumed to reconfigure the square array to avoid PEs found defective after fabrication, arrays in the range 20×20 to 30×30 are possible. The possibility to exploit at best this sort of devices in a modular machine could prove particularly positive.

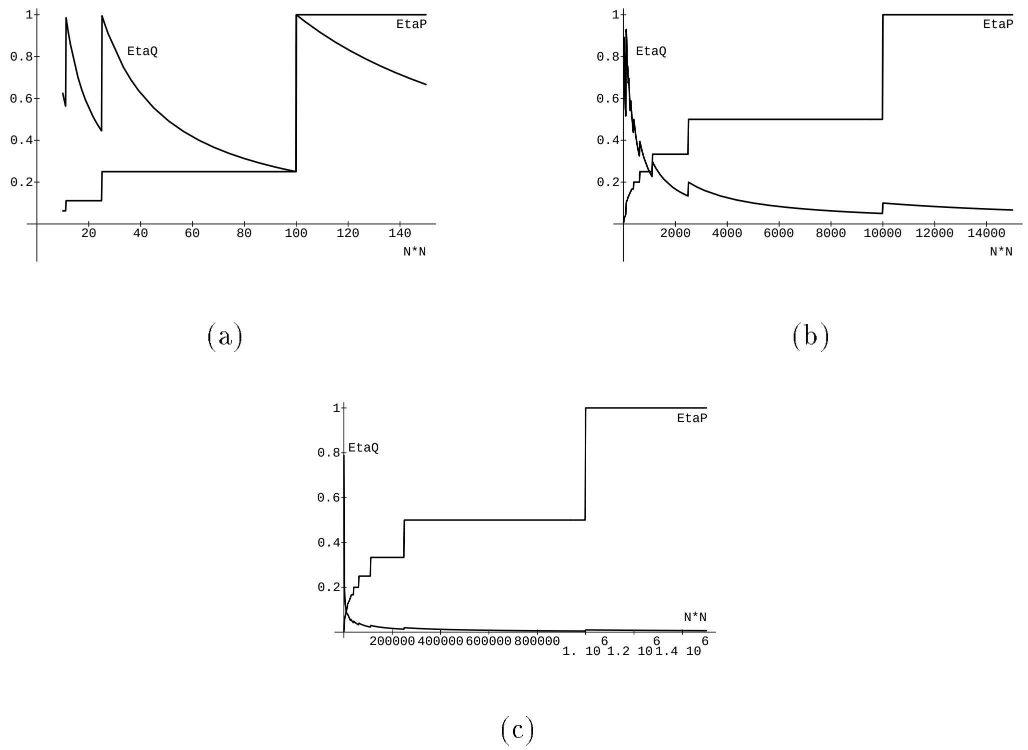

Figure 18. GENES IV. Plots of η_P and η_Q in function of the number of PEs (N^2) in the array. η_P is an increasing function of the array size, eventually reaching one when $N = \max(n, m)$. The three plots correspond to different matrix sizes: (a) $n = 10$ and $m = 10$; (b) $n = 10$ and $m = 100$; (c) $n = 10$ and $m = 1000$.

2.3.2 J. H. Chung *et al.* Array

In [Chu92], an architecture somewhat similar to GENES H8 ([Leh91]) is presented. In contrast to the latter—dedicated to the Hopfield and Kohonen networks—the Chung architecture is devoted to the back-propagation algorithm.

Figure 19 (from [Chu92]) shows a sample array for a (5-3-2) 2-layer network. Each processing element (shown as a square containing a weight) performs the operations shown in figure 20. The θf element output is the value of the non-linear function of the input, while the g elements contain a FIFO and produce the value of the derivative of the non-linear function. Basically, the idea is to have an array for each layer of the network, each producing the output and feeding it to the input of the next layer. At the output of the last layer, the difference with the desired output is computed and fed back—on a different (and orthogonal) path—to the arrays; there the error vectors get multiplied by the transposes of the weight matrixes and back-propagated. Periodically the feedforward pipeline is stalled and the weights are updated using a special processing element operation.

Being the systolic structure essentially the same, many similarities can be found with GENES IV (see §2.3.1). Among them, it is worth underline that the product by $\mathbf{W}^{\mathrm{T}}$ instead of $\mathbf{W}$ is performed in essentially the same way as it is in GENES IV. In contrast to the GENES approach, here both products are performed at the same time by replicating the multiplier/adder inside the processing element. Compared to the GENES IV solution, this is a clever idea as it avoids—

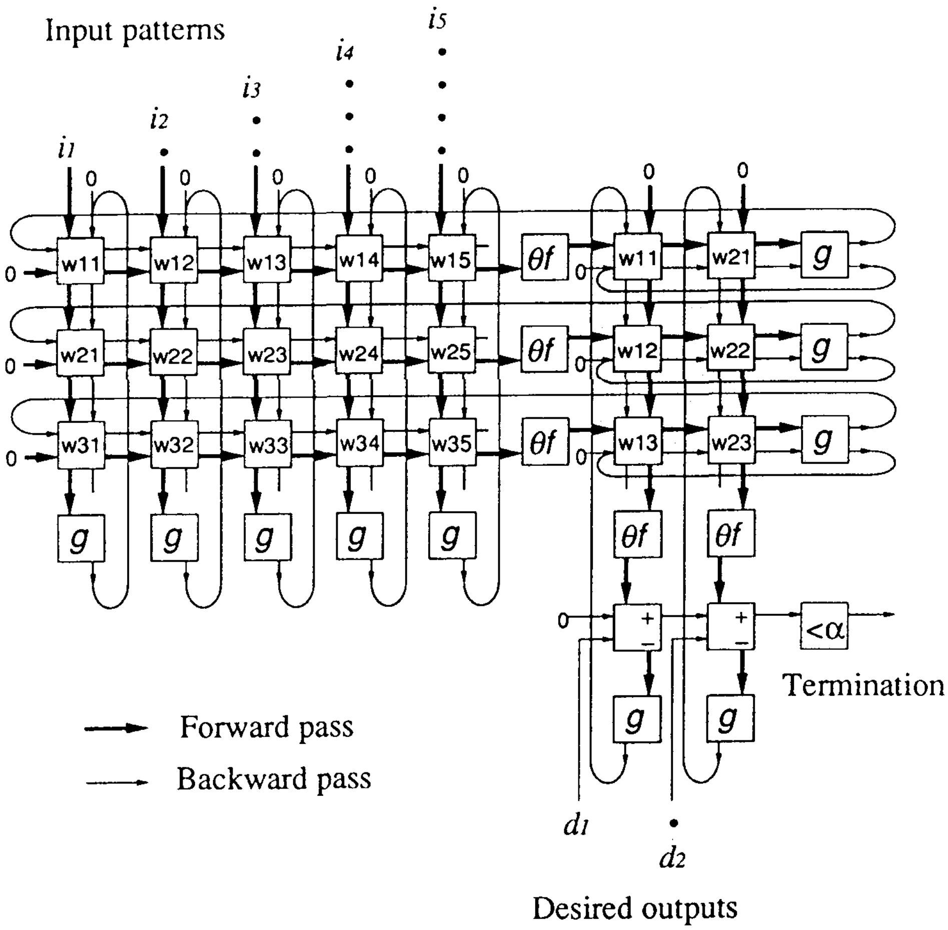

Figure 19. CHUNG *et al.* Systolic Array.

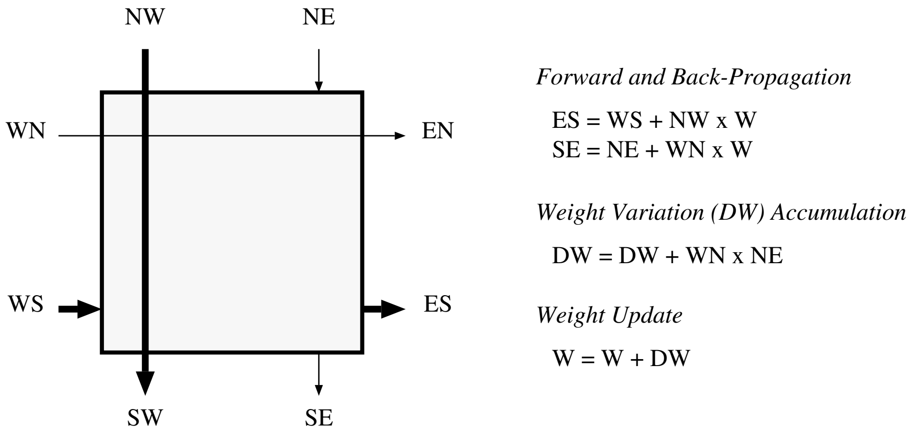

Figure 20. CHUNG *et al.* Systolic Array Processing Element.

as in GENES—to alternatively use the matrix for the forward and back-propagation phase, but it doubles the number of active arithmetic units in the pipeline (and, finally, the processor complexity). Unluckily, this second multiplier/adder is only useful for the back-propagation algorithm and thus probably not efficiently useful for other NN paradigms. In fact, the biggest difference with the GENES architecture consists in the fact that the Chung design is not aimed at housing generic networks or models:

- The described system is dedicated (1) to the back-propagation algorithm and (2) to a fixed network configuration.
- The architecture—being physically modelled on a specific network size—does not account for a possibility of virtualisation.

2.4 Conclusions

The examples described in this section are far from being exhaustive, also limiting to digital multi-model systems. Nevertheless, it can be noted that the number of different system structures is relatively small and they are well represented here. An observation comes to mind about the virtual network support: While some of the architectures are decidedly targeted at a one-to-one processor to synapse or neuron implementation (e.g., the Chung array), other very general machines (the SIEMENS being the most important example) seem to be penalized by missing a virtual network support. In general, this is often the case for ring architectures, with the exception of the Kung and Vlontzos one ([VLO91]). In this case the authors acknowledge the importance of the problem and plan future work in the direction that they define as *"the most crucial one for the performance of a neural network simulator"*. This view corresponds to the fact that one-to-one implementations are economically unacceptable or technically unfeasible for real-world problems and digital implementations. This requires further work on the domain and justifies the study of elementary techniques for virtual matrix support that will be performed on some fundamental architectures in the next section.

3 Analysis of Architecture Classes

Having considered a group of example architectures, it is now possible to begin to compare the different architectural classes. For this purpose—and somewhat in the way [Ful77] compared μP's—*absolute criteria* and *quantitative criteria* can be introduced. To the first class belong some criteria which are often hardly measurable and corresponding to broad and fundamental features of the architecture. They are in the form of boolean indicators assessing the availability of a given feature. It is fair to mention that some absolute criteria are even somewhat subjective or depend on the perception of the system that the author gained by reading the relevant papers. The quantitative criteria are necessary to correct this sort of subjectivity and to assess, when possible, performances on the basis of evaluable values. This allows a finer-grain comparison among different systems and possibly the detection of the key advantages of each of the approaches.

A first section will introduce the absolute criteria and their rationale. The criteria will be used directly on the examples described in §2. Basically, the criteria will be used to screen-out some of the architectures which do not comply to the general ideas of a neuro-computer as depicted in the introduction and at the same time they allow to resume some of the characteristics of the examples seen so far. Next some classes of architectures will be defined, grouping the various implementations that share similar basic concepts. Each of the architecture will be briefly described and some features, as the flow of the data, idealized to make the computation of the quantitative parameters possible. §3.3 presents these parameters and §3.4 produces comparative tables of the values measured on a sample problem. Some comments to these values are made in §3.5 and a critical picture of each architecture is attempted. Finally, in §3.6, some general conclusions are drawn on the chosen analysis approach.

3.1 Absolute Criteria

Not all the architectures handle all the functions which are felt as needed for mapping general neural networks. Absolute criteria should give this kind of perspective, eventually allowing for the exclusion of some architectures from further analysis. In this broad sense, important absolute criteria have already been implicitly introduced at the beginning of this document by excluding from the comparison analog implementations because of the low (or null) degree of programmability.

The following absolute criteria are now considered:

1. **Virtual Size Matrix Support** (V). By this it is meant that operations on matrixes larger than that for which the physical array is conceived can be mapped on the hardware. Swapping of weights, if it cannot be performed in the background during the computations, is considered as a feature that prevents virtual matrixes to be supported. The criterion is considered satisfied when the authors address directly the problem explaining the techniques used to map the virtual problem on the machine. The absence of such indication is independent of possible generalizations applicable to the same architecture.

2. **Multi-Model** (M). The criterion establishes if an architecture has been conceived with more than one NN paradigm in mind.

3. **Programmability** (P). It is a feature hard to define formally, thus making this a somewhat subjective evaluation. The criterion should establish systems which are multi-model but not restricted to a small, predetermined number of paradigms. This implies that the

Design	V	M	P	T	O	D
CNAPS	○	●	●	○	●	●
Hitachi (w/o BP)	○	○	○	○	○	○
Hitachi (with BP)	○	○	○	○	●	●
Lneuro 1.0	●	○	○	○	●	○
Sprint	○	○	○	●	●	●
TNP	○	●	●	●	●	●
SNAP	○	●	●	●	●	●
SYNAPSE-X	○	●	○	●	●	●
GENES IV	●	●	○	●	●	●
Chung *et al.*	○	○	○	●	●	●

Table 2. Absolute Criteria. The first column indicates the design, the others contain each of the absolute criteria under the symbols introduced in the description of these parameters. Empty circles mean that the criterion is not satisfied while filled ones means a positive assessment.

processing elements are not strictly dedicated to a few arithmetic operations conceived by the designers to be used in a rather strict sequence, but rather allowing some software freedom. Features that make an architecture programmable in the sense interesting here are, for instance, register redundancy or register addressability in the instruction code.

4. **Transpose Matrix Support** (T). It indicates that the proposed hardware allows indifferently for the use of the stored matrix in its direct or transpose form. The matrix should be contained in the hardware only once, for the criterion to be acknowledged. Again, background swapping of weight is considered permissible.

5. **Outer Product Support** (O). An architecture adapted to the Hebb learning rule (and thus to back-propagation) should perform efficiently a computation of the form $\vec{\mathbf{v_1}} \cdot \vec{\mathbf{v_2}}^{\mathrm{T}}$.

6. **Term-to-Term Product Support** (D). Similarly, support for the operation $p_i = a_i \cdot b_i$ on two vectors $\vec{\mathbf{a}}$ and $\vec{\mathbf{b}}$ should be available.

Table 2 shows the results of the application of the criteria to the examples described in §2. The following architectures are discarded from further analysis for the mentioned reason:

- **Hitachi.** Both designs are based on a broadcast bus. As already mentioned, these are more interesting because of the implementation than for the architecture itself. The architectural style is better represented by the more general-purpose CNAPS design.

- **Lneuro 1,0.** It cannot perform the term-to-term product of two vectors, thus relying for this operation on the host processor. This severely hinders the performance by serializing a necessary operation and prevents any computation of back-propagation performance.

- **Chung** *et al.* As mentioned when discussing the architecture, it is a design essentially dedicated to a given network size and to a given model. The first element implies that it fails to comply to criterion V (note that the weights are residents in the processors). This choice produces an efficient architecture highly adapted to the problem but consequently more comparable—in terms of practical applications—to some non-programmable analog designs.

3.2 Architectural Classes

To better apply the quantitative criteria developed in the sequel, the systems studied in §2 will be grouped by their data flow structure. A synthetic picture of each of the classes obtained will be given in this section. In some cases, their characteristics are expressed through variables, drawing a picture further away from implementation details and constraints. This implies that some simplifications will be, at times, implicitly needed to allow an easier computation of the parameters. The example architectures of each class will be mentioned. Usually reading the referenced section on the example architecture will be needed to understand the data flow. Appendix A, detailing the quantitative criteria computation, briefly explains, in critical cases, the data flow used to perform the computation, thus clarifying potentially obscure points.

One important remark should be done about the degree of idealization of the described systems: The present document is concerned with the assessment of the architecture performance and with the characteristics of the relative processing element, hence resources external to the array of processors are not detailed. This implies that:

- Communication channels drawn on the figures must support the transfer of a word per cycle. This applies also to external memories, which are always supposed to sustain a throughput of one word red or written every cycle.
- External resources, such as memories or FIFOs, are of unlimited size. (Better, they are never the elements limiting the performance or capabilities of the architecture.) This is evident if the use of a machine for a problem larger than the physical array is considered: The larger weight matrix must be stored somewhere and the only way to do so is to consider the external weight matrix as being larger. In a sense, this infinite availability of resources is often justified by the regularity and modularity of the external hardware.

It is at least fair to admit that the design of the external hardware is not trivial at all and often the optimal results—under given technology constraints—significantly hinder the performance of an architecture. Nevertheless, the rationale of this document is that a good selection of an architecture is the first, necessary step in the design of a system. The feel is that loss of performance by an inaccurate choice of the architecture may not be compensated later by an accurate design of the hardware external to the processing array.

Another remark concerns the arithmetic precision: A typical parameter considered in neurocomputer comparison is the number of bits used for storing all the different types of variables. Usually the problem, if considered in all its implications, is rather complex, especially for fixed-point machines (as most of the considered designs are) where improper choices may compromise the whole practical performance. This aspect is not considered here because at an architectural level it can be supposed that any precision can be later implemented or, more realistically, the possible range of well-chosen precisions does not severely affect the performance (e.g., in terms of throughput) of the machine.

3.2.1 Broadcast Bus Architecture (BBA)

Figure 21 shows the Broadcast Bus Architecture. An important distinctive element is the weight register file internal to the processor. None of the other architectures considered has this feature. It is implemented in this basic form (with minor variations) in the CNAPS system (see §2.1.1) and in the HITACHI ones (see §2.1.2). The former is by far the most general system of the class and, under several aspects, one of the more interesting neuro-computers for modularity and versatility. Two versions of the BBA are considered for quantitative criteria

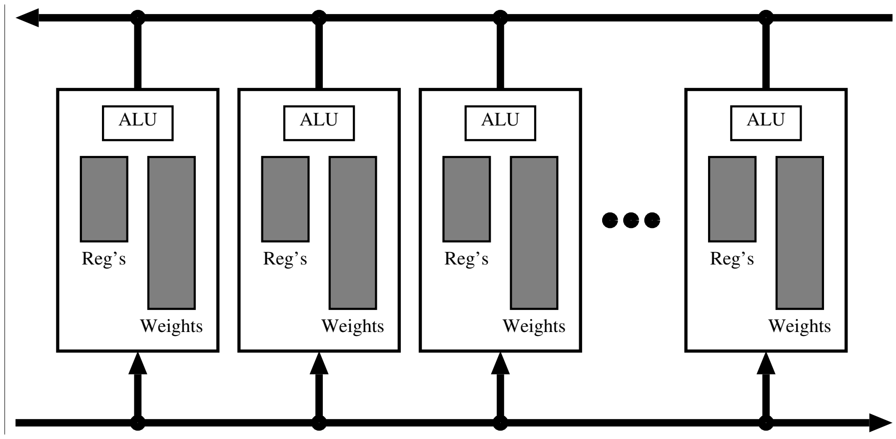

Figure 21. BBA. Broadcast Bus Architecture

calculation. Actually, the BBA is strongly penalized if the weights have to be swapped during the calculation and this is the case for matrix transposition. Thus the storage is minimized for optimal throughput on the direct matrix in system BBA/I, while BBA/II is a system minimized for optimal throughput in both direct matrix and transpose multiplication. Moreover, the computation on BBA/II is done considering an epoch length of $E^{\star}$. Appendix A.1 details the differences between the two systems.

3.2.2 Systolic Ring Architecture (SRA)

Figure 22 shows the Systolic Ring Architecture. It is one of the most popular architectures, obtained from a classic matrix-vector multiplier implementation. Virtual array support is very limited. In the idealized version presented here, it is admitted (as mentioned before) that not only the weight memory but also the FIFO length can be extended at will. This feature is essential to virtual array computation. Among the examples cited, the SPRINT implementation (see §2.2.1) can be referenced for information on the data flow.

One important remark should be done with respect to the processing element assignment. Figure 7(a) and 7(b) show that the architecture can be equally well used by storing the $m \times n$ matrix row-wise or column-wise. It is easily verified that if the number of processors is $\min(m, n)$, the length of the ring should be of $\max(m, n)$ elements. This is the function of the FIFO. Conversely, if the number of processors is $\max(m, n)$, which could improve the parallelism exploitation, the length of the ring should be of $\min(m, n)$ elements. Being physically impossible, this implies that—if batch processing of the input vectors is not considered—$\max(m, n) - \min(m, n)$ processors must be inactive at each cycle. This would reduce the parallelization efficiency but should improve the throughput and the parallelism exploitation. Consequently, both cases presenting interesting features, the evaluation of the performances will be performed in two configurations: Type **MIN** contains $\min(m, n)$ processors whilst type **MAX** has $\max(m, n)$.

3.2.3 Systolic Ring Architecture with Global Bus (SRAGB)

Figure 23 shows the Systolic Ring Architecture with Global Bus. It is obtained from the SRA with the addition of a global interconnection bus. This allows any processor to send a data word

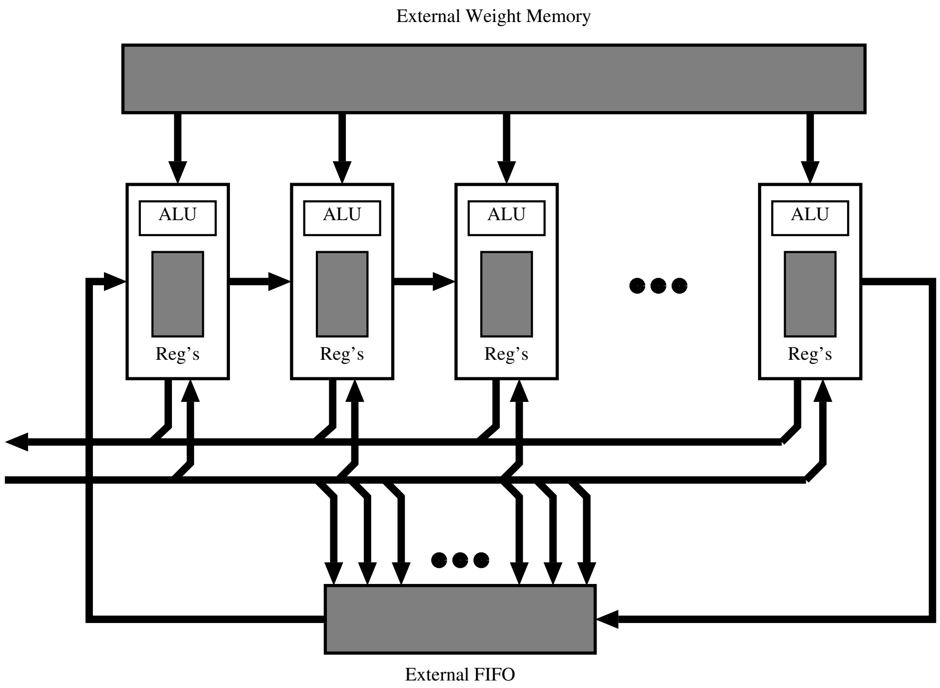

Figure 22. SRA. Systolic Ring Architecture

to any other processor in a single cycle. The purpose of this is essentially to allow the system to add efficiently partial terms of sums dispersed over many processors. This could be useful in the SRA/MAX to allow a better parallelization by allowing many partial sum to develop at the same time. The HNC system (see §2.2.3) is an example of this architecture.

3.2.4 Multiple Systolic Ring Architecture (MSRA)

The *Multiple Systolic Ring Architecture (MSRA)* is composed by several SRA/MIN in parallel, fed by the same weight memories. The structure is essentially the same of the SRA/MIN. The purpose is simply to enhance the throughput by processing more prototypes or input data at the same time. The class is somewhat represented in the previous section by the SIEMENS system when the MA16 circuits are arranged in a 2D array (see §2.2.4)[7]. The remark done on the matrix arrangement in the SRA/MIN applies unmodified in the case of the MSRA.

3.2.5 Systolic Square Array Architecture (SSAA)

Figure 24 shows the Systolic Square Array Architecture. It results from another classic architecture for matrix-vector multiplication. It has been shown that the same data flows can be used to perform more general operations. On one side it appears as one of the most promising architectures by allowing a parallelism exploitation efficiency (see §2.3.1) of 100 %. On

[7]In §3.2.6 the MA16 is also given as an example of SRMPA. In fact, the SIEMENS architecture is an hybrid architecture possessing several basic features at the same time. More exactly it can be affirmed that adding a second dimension to the MA16 network is expected to produce the same results that a MSRA displays when compared to an SRA/MIN.

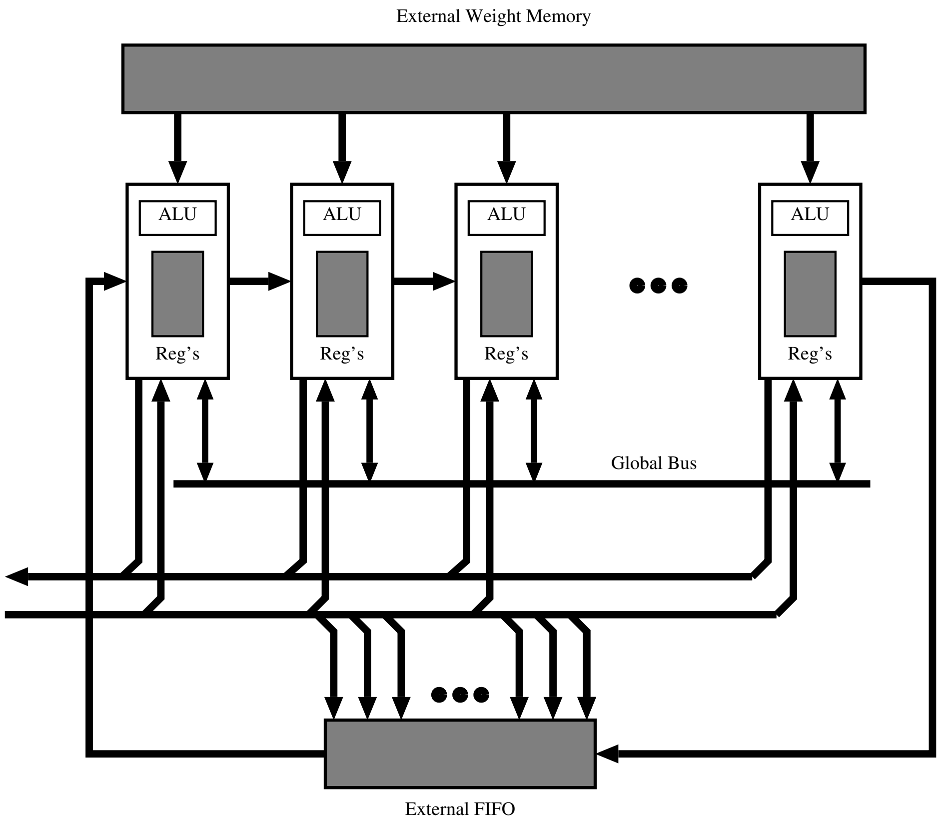

Figure 23. SRAGB. Systolic Ring Architecture with Global Bus

the other side, it encounters several problems in exploiting the processing elements when large arrays are integrated and networks with very different input/output counts are considered. A typical multi-model SSAA is the MANTRA I system (see §2.3.1). Note that this architecture requires external delays to feed the array with elements from the same vector in subsequent cycles (*diagonalization*). In the case of GENES IV, these are built inside the processors.

3.2.6 Systolic Ring of Matrix Processors Architecture (SRMPA)

This architecture corresponds exactly to the SRA/MIN but all the data circulation and handling is performed on matrixes instead of scalars. The use of matrixes for quantities like $\vec{\mathbf{x}}$ or $\vec{\mathbf{p}}$ implies the introduction of an epoch size. These matrixes are built by concatenating several successive input vectors. The size of the side of the square matrixes handled by the processors will be denoted by N. Once more, the remark done on the matrix arrangment (column-wise or row-wise) in the SRA applies equally well to both types of SRMPA. Two types of SRMPA will be distinguished:

- **Type I: 1D Matrix Processors**. In this type of architecture, each matrix processor is a 1D systolic array of scalar processors, as an SRA or a similar structure. An example of this structure is the SIEMENS system composed of Elementary Chains processing 4×4 matrixes. The MA16 Elementary Chain is used to compute the quantitative criteria.

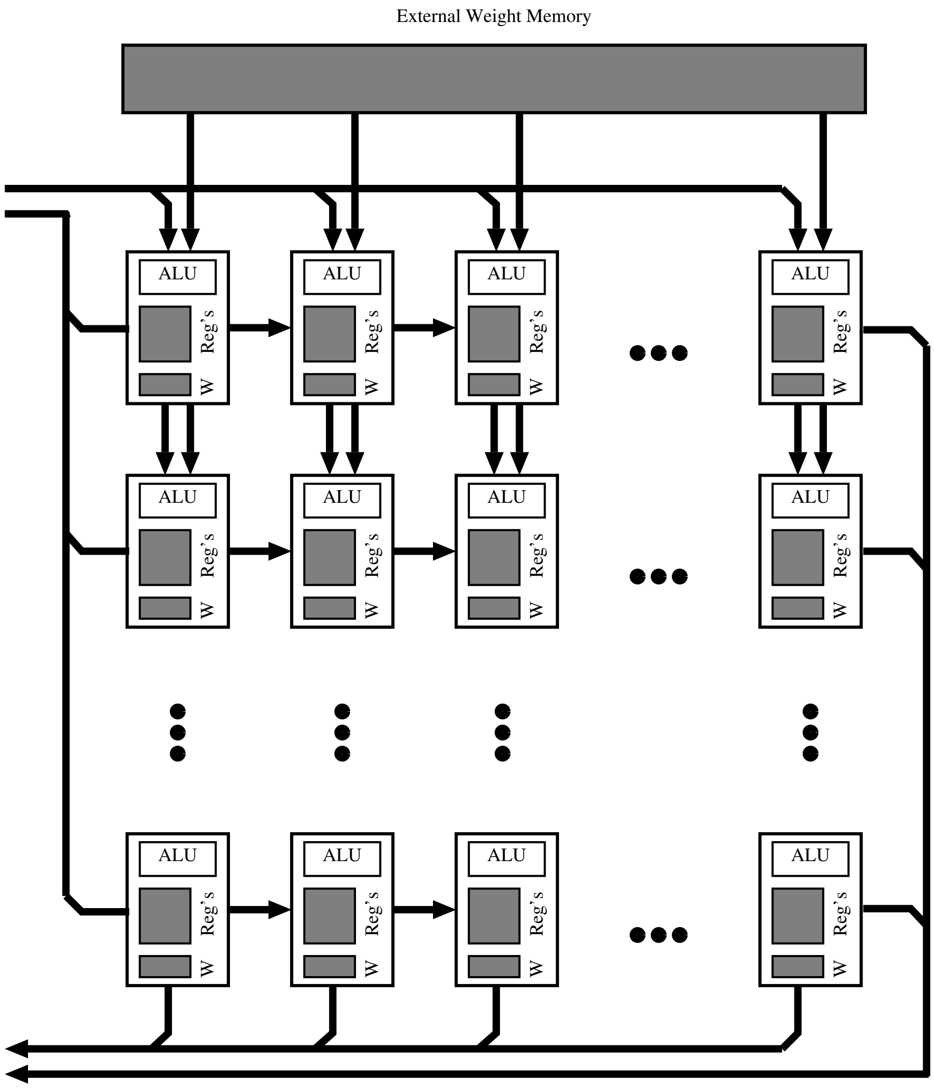

Figure 24. SSAA. Systolic Square Array Architecture

- **Type II: 2D Matrix Processors.** In Type II, conversely, the matrix processor is a 2D systolic array. No system of this type is known in literature. The following analysis will suppose an SSAA for the matrix processors.

3.3 Quantitative Criteria

Any set of quantitative criteria needs to consider a specific problem (e.g., matrix-vector multiplication) to allow the evaluation of performance indexes. Unfortunately the performance of an implementation of a given architecture depends dramatically on the characteristic size of the problem and on its ratio to the system size. The following approach is chosen to minimize these effects:

- The matrix-vector multiplication problem is considered as the sample operation to perform. Most of the architectures could implement other matrix-vector operations with the same efficiency simply by adding the necessary operations to the arithmetic unit (e.g., the use of the same data flow for Euclidean distance computation is demonstrated in GENES IV). The problem is chosen in parametric form, that is the weight matrix $\mathbf{W}$ has size $m \times n$ and the input vector $\vec{\mathbf{x}}$ has size n. The result, or potential $\vec{\mathbf{p}}$, is consequently a vector of size m.

- Given the problem, a system is hypothesized with the given architecture. All the parameters (number of processor, storage, etc.) are optimized to maximize the throughput for the given problem.

The idea is to put the different architecture in the same *optimal* condition with respect to the basic problem. Under these conditions a number of characteristic values are computed. A first group (from # 1 to # 5) measures some characteristic quantities of the architecture. The others are intended to assess the performance of each system when dealing with the basic problem and with some *significative* case showing the adaptability to other operations or network sizes. The following list details the various parameters and briefly discusses their rationale:

1. **Number of Processors** (N_P). Counts the number of processors (usually identified with the arithmetic unit performing the multiply/add operation) constituting the system.

2. **Weight and Data inputs** (I_W and I_x). Counts the number of word inputs for the weights and for the input vector. Serial communication systems have their inputs and outputs considered as parallel, the serial I/O being considered only as a way to reduce the otherwise unacceptable connectivity down to a reasonable level and producing a fixed reduction in throughput (e.g., a factor 40 in GENES IV, see §2.3.1). In contrast, the number of communication channels may have an important impact (dependent on the size of the problem) on the throughput of the system (see §2.1.1, as an example).

3. **Result outputs** (O_p). Counts the number of word outputs for the result vector.

4. **Data Storage per Processor** (S_P). Counts the total number of bits of data storage in each processor per bit of precision in the weight register. The width of the weight register is considered as an approximate measure of the arithmetic precision used. The storage includes internal weight register files or internal replicas of the data being processed. On the contrary, it does not include the storage required for the control/microsequencing of the processor, neither the internal temporary storage of the arithmetic unit. Of course,

sometimes it is hard to distinguish between the internal replicas needed to reduce the external connectivity and those required—in addition, perhaps—for the arithmetic handling. Therefore, this value is more a qualitative index than an absolutely precise measure. It is a parameter that is often desired to keep to a minimum as internal storage is typically expensive while external storage may take advantage of cheap commercial devices (both DRAMs and SRAMs) at much lower cost per bit.

5. **Epoch Size** (E). Pipelined architectures require a minimum (or optimal) number of successive repetition of each operation on new data to keep the pipeline full and exploit the hardware at a maximum. This is due to the fact that, being the latency greater than the inverse of the throughput, new data can be fed before the result of the first operation is ready and thus available for subsequent calculations (e.g., to propagate through the next layer or to back-propagate the error). The number of vector processed together is called *Epoch* or *Batch*. The epoch size simply counts the number of vector in an epoch.

6. **Latency** (λ). Counts the number of elementary cycles between the input vector presentation and the output appearance. The *elementary cycle* is assumed to be the time required by a processor to multiply two input data and add the result to a third input data. The weight matrixes are, in many architectures, resident in the system. In these cases the weight loading time is not included in the latency or in the other parameters described in the sequel. On the contrary, the time to swap the weights for virtual network support is, when present, taken into account as well as the time to reset the system to its initial state. In general, the latency is—especially in the case of systolic systems—less interesting than the throughput (see below); nevertheless it can be fundamental in certain applications (interactive systems) and long latencies are often reflected in long epoch sizes.

7. **Throughput** (θ). The throughput represents the number of complete matrix-vector products computed per elementary cycle. In most cases it is the key figure of merit for the applications (typically determinant to the MCPS and MCUPS measurements).

8. **Parallelization Efficiency** (η_Q). It is the figure of merit already introduced when discussing GENES IV in §2.3.1. It shows how well the parallelization scheme performs. Ideally the throughput should increase linearly with the number of processors: In such optimal cases the parallelization efficiency is a constant. If this is not the case (e.g., the parallelization efficiency is a decreasing function of some physical characteristic size of the system), this indicates limits to the scalability of an architecture. The parallelization efficiency is defined as

$$\eta_Q = \frac{\text{Throughput of the System}}{\text{\# of Processing Elements} \cdot \text{Throughput of a Single Processor}}.$$

Considering that the matrix-vector multiplication throughput of a single processor is the ratio between the multiply-add single processor throughput (one, by definition of elementary cycle) and the number of multiply-add operations in the problem ($n \cdot m$), η_Q can be re-expressed as

$$\eta_Q = n \cdot m \cdot \frac{\text{Throughput of the System}}{\text{\# of Processing Elements}}.$$

9. **Transposition Performance** (ξ_T). Even though the basic problem considered is the matrix-vector multiplication, an architecture conceived for NN algorithms—and namely

for the back-propagation algorithm—should perform efficiently the matrix-vector product using the transpose of the weights. The transposition performance can be defined as

$$\xi_T = \frac{\text{Throughput on the Transpose}}{\text{Normal Throughput}}.$$

It is ideally equal to one (balanced performance between the forward and backward phase), but can be any positive number. Values of ξ_T greater than one mean a good transpose multiplication performance but may show a less than ideal forward performance.

10. **Reconfiguration Performance** (ξ_R). The ability of a given architecture to deal with problems of the same arithmetic complexity (i.e., number of elementary operations) but of different matrix aspect ratio is an interesting indicator of the versatility of the communication network between processors. The reconfiguration performance can be defined as

$$\xi_R = \frac{\text{Throughput on the Rearranged Problem}}{\text{Normal Throughput}}.$$

The *rearranged problem* is still a matrix-vector multiplication but with the matrix of size $k_R m \times n / k_R$. (To simplify the calculations, it is assumed—for the purpose of this criteria—that $n = \alpha k_R$ with $\alpha, k_R \in \mathbb{N}$.) The reconfiguration performance is ideally equal to one (perfect reconfigurability of the physical network), but can be any positive number. As in the case of the transposition performance, values of ξ_R greater than one mean a good rearranged matrix multiplication performance but may show a less than ideal forward throughput.

11. **Virtualization Performance** (ξ_V). Similarly, it is interesting to evaluate the performance of an architecture when dealing with problems of different size from the basic one considered. It is hardly possible—for a general purpose experimental neuro-computer—to have enough processors to parallelize at best any reasonable network. A measure of the capability of handling matrixes of larger size can be assessed by measuring the virtualization performance, defined as

$$\xi_V = \frac{\text{Throughput on the Virtual Size Problem}}{1/{k_V}^2 \cdot \text{Normal Throughput}}.$$

The *virtual size problem* is still a matrix-vector multiplication but with the matrix of size $k_V m \times k_V n$, with $k_V \in \mathbb{N}$. The ideal result is typically one (i.e., performance reduced of the factor k_V^2 which represents the increase in arithmetic complexity of the problem). Lower values indicate increasingly bad performances while values above one are welcome but often arising from bad parallelization efficiencies on the basic problem.

3.4 Analysis Results

Tables 3, 4 and 5 illustrate the values calculated for the various architectures. Detail of the computations can be found in appendix A. The results are usually functions of m, n, k_R and k_V and of the characteristic parameters of the architectures ($E^\star$ and N).

Table 3 presents a first group of parameters, notably the physical characteristics of each architecture. Table 4 shows the basic performances on the $m \times n$ matrix, for each of the various architectures considered. Table 5 shows the other performances.

Architecture	N_P	I_W	I_x	O_p	S_P
BBA/I	m	1		1	$\simeq n$
BBA/II	$\max(m, n)$	1		1	$\simeq m + n$
SRA/MIN	$\min(m, n)$	$\min(m, n)$	n	m	2–3
SRA/MAX	$\max(m, n)$	$\max(m, n)$	n	m	2–3
SRAGB	$\max(m, n)$	$\max(m, n)$	n	m	2–3
MSRA	$N \cdot \min(m, n)$	$\min(m, n)$	$N \cdot n$	$N \cdot m$	2–3
SSAA	$\max^2(m, n)$	n	n	m	7
SRMPA/I	$\lceil \min(m, n)/N \rceil \cdot N$	$\lceil \min(m, n)/N \rceil$	$\lceil n/N \rceil$	$\lceil m/N \rceil$	6
SRMPA/II	$\lceil \min(m, n)/N \rceil \cdot N^2$	$\min(m, n)$	n	m	7

Table 3. QUANTITATIVE CRITERIA. The first column indicates the architecture, the others contain processor count, inputs and outputs and data-storage under the symbols introduced in the description of these parameters.

3.5 Discussion of the Results

The shown tables allow some general comments on the different architectures. The BBA architecture—at least in its stored-weight version presented here—is not an architecture specially adapted to a general-purpose neuro-computer: It shares most of the SRA performances with the notable exception of an expectedly bad virtualization performance ($\eta_V = 1/(m+1)$). The evident advantage of this architecture is the easy integrability (no memory interface, high modularity, easy high processing element count). In a sense, it seems more an ideal architecture for complex on-field applications than for a neuro-computer.

SRA/MIN and MSRA have good transposition performance ($\xi_T = 1$) and are well applicable to problems larger than the physical matrix ($\xi_V = 1$). The reconfigurability is relatively low for some important combinations of m, n and k_R ($\xi_R < 1$ for $k_R > n/m$, that is, for aspect ratioes further away from 1 than the transpose). If a good reconfigurability can be attained in many cases, this is essentially due to the fact that one of the dimensions of the problem is physically mapped on the hardware (one processor per column or row) while the other is time multiplexed and thus under the programmer control. The biggest problem with the SRA/MIN is the fact of using a simple mapping algorithm which is most efficient on $\min(m, n)$ processors. This fact implies a low exploitation of the problem parallelism for either a low n or m value, that is for highly non-square matrixes.

If a better exploitation of the parallelism is envisaged, SRA/MAX can be considered. Unfortunately it shows a strong inefficiency for $n \gg m$. However, acceptable ξ_R and ξ_V are obtained. The low η_Q is in the same order of the SSAA value (see below), thus making the SRA/MAX only moderately interesting. Alternatively, the introduction of epochs (in the order of $\max(m, n)/\min(m, n)$ elements) may allow to improve the parallelization on $\max(m, n)$ processors but at the expenses of potentially very high E, which can be annoying in many applications. It can be expected that this could also improve the reconfiguration efficiency. Once more, the epoch length problem is common to the SSAA paradigm, whose other advantages may make it preferable.

A different—and in principle more promising approach—would be to study more efficient

Architecture	λ	θ	η_Q
BBA/I	n	$1/n$	1
BBA/II	n	$1/n$	$m/\max(m,n)$
SRA/MIN	n	$1/\max(m,n)$	1
SRA/MAX	n	$1/n$	$m/\max(m,n)$
SRAGB	$\min(m,n)+\left\lceil\frac{n}{\min(m,n)}-1\right\rceil$	$\frac{1}{\lceil\max(m,n)/m\rceil\cdot\min(m,n)}$	$\frac{1}{\lceil\max(m,n)/m\rceil}$
MSRA	n	$N/\max(m,n)$	1
SSAA	$2\max(m,n)$	1	$\min(m,n)/\max(m,n)$
SRMPA/I	$\lceil n/N\rceil\cdot N^2$	$\frac{1}{\lceil\max(m,n)/N\rceil\cdot N}$	$\frac{n\cdot m}{\lceil n/N\rceil\lceil m/N\rceil N^2}$
SRMPA/II	$\lceil n/N\rceil\cdot 2N$	$\frac{1}{\lceil\max(m,n)/N\rceil}$	$\frac{n\cdot m}{\lceil n/N\rceil\lceil m/N\rceil N^2}$

Table 4. Quantitative Criteria. The first column indicates the architecture, the others contain the latency, the throughput and the parallelization efficiency under the symbols introduced in the description of these parameters.

Architecture	E	ξ_T	ξ_R	ξ_V
BBA/I	1	$\frac{n}{m\lceil n/m\rceil+2mn}$	1	$1/(m+1)$
BBA/II	$E^\star$	n/m	$\max(m,n)/m$	$E^\star/(m+E^\star)$
SRA/MIN	1	1	$\frac{\max(m,n)}{k_R\max(m,n/k_R)}$	1
SRA/MAX	1	n/m	$k_R/\left\lceil\frac{k_R m}{\max(m,n)}\right\rceil$ $\left(k_R>\sqrt{\frac{n}{m}}\right)$	$k_V/\left\lceil\frac{k_V\min(m,n)}{n}\right\rceil$
SRGBG	1	$\frac{\lceil\max(m,n)/m\rceil}{\lceil\max(m,n)/n\rceil}$	$\frac{k_R\frac{m}{\max(m,n)}\left\lceil\frac{\max(m,n)}{m}\right\rceil}{\left\lceil\frac{k_R m}{\max(m,n)}\right\rceil}$ $\left(k_R>\sqrt{\frac{n}{m}}\right)$	$\left\lceil\frac{n}{min(m,n)}\right\rceil$ $(k_V>\frac{n}{m})$
MSRA	N	1	$\frac{\max(m,n)}{k_R\max(m,n/k_R)}$	1
SSAA	$2\max(m,n)$	1	$1/\left(\left\lceil\frac{k_R m}{\max(m,n)}\right\rceil\left\lceil\frac{n/k_R}{\max(m,n)}\right\rceil\right)$	$\frac{k_V}{\left\lceil k_V\cdot\frac{\min(m,n)}{\max(m,n)}\right\rceil}\geq 1$
SRMPA/I	N	1	$\frac{\lceil\max(m,n)/N\rceil}{k_R\lceil\max(m,n/k_R)/N\rceil}$	$\frac{k_V\lceil\max(m,n)/N\rceil}{\lceil k_V\cdot\max(m,n)/N\rceil}$
SRMPA/II	$2N$	1	$\frac{\lceil\max(m,n)/N\rceil}{k_R\lceil\max(m,n/k_R)/N\rceil}$	$\frac{k_V\lceil\max(m,n)/N\rceil}{\lceil k_V\cdot\max(m,n)/N\rceil}$

Table 5. Quantitative Criteria. The first column indicates the architecture, the others contain the characteristic epoch size, the transposition performance, the reconfiguration performance and the virtualization performance under the symbols introduced in the description of these parameters.

mapping algorithms that, for instance, make use of more processors (e.g., $\max(m, n)$) rearranged for the purpose as a binary adder tree to parallelize and improve the mapping of SRA/MAX. The SRAGB can be viewed as an effort in this direction by allowing a more efficient addition of the partial sums. Unfortunately the throughput shows that the number and the arrangement of the partial sums in the matrix-vector multiplication operation makes the global bus inefficient (it is always $\theta_{SRAGB} \leq \theta_{SRA/MAX}$, being, when $n > m$, $\frac{1}{\lceil n/m \rceil m} \leq \frac{1}{n}$).

When high epoch lengths are acceptable, the SSAA model emerges as the most effective architecture. Unfortunately, as it has already been remarked, it is however far from efficient if highly non-square matrixes and networks self-contained in the hardware are considered. On the positive side it should be noted that the virtualization efficiency is always above unity due to the possibility of modifying the operation mode of the array without lost cycles (systolic instruction flow, see [VIR92]) and to the background swapping of the matrixes. Similarly the transposition performance is always one (the fact has strong implications with the extremely low parallelization efficiency for highly non-square matrixes).

Considering now mixed architectures, it can be noted that the SRMPA/I has performances almost identical to those of the SRA/MIN. This was somewhat predictable, the SRMPA/I replicating the same type of network at the two hierarchical levels. The reason for implementing this architecture (as in the case of SYNAPSE-X) resides more on technological considerations (reduced connectivity, compare I_W, I_x and O_p among SRA/MIN and SRMPA/I) than on throughput enhancements. On the contrary, the SRMPA/II could be expected to have some interesting features on this side: The throughput is enhanced by a factor N, which is an important factor with respect to all the other 1D architectures. In fact, the MSRA shares this feature but requires the weight update phase to be serialized because all the rings produce the weight updates at the same time and all of these must be added to the weight matrix before proceeding. Another interesting feature is that the epoch length is modelled on the matrix processor size and thus can be kept low to acceptable levels. It must be emphasized that—if a better parallelization is envisaged—the same considerations done on the SRA/MIN apply; it must be stressed that if no better mapping algorithms can be found, the epoch length may then grow towards the levels of the SSAA model, making the SRMPA/II much less interesting.

3.6 Analysis Conclusions

This type of quantitative analysis—even if still at an early stage—seems to provide valuable and objective information on the advantages and defects of the different possible architectures. This allows an architecture designer to identify critical research directions which require further development. For example, in the case of an original architecure (SRMPA/II), it allows to quantify the criticity of the mapping problem whose solution may make this architecture interesting.

A problem can be identified in the architectural classes as defined in this document: A better idea would be to develop some sets of mutually exclusive and consistent features (e.g., topology style, weight storage style, scalar or matrix processing elements). Then classes can be formed by combining features from the various sets. This approach could allow to (1) verify the impact of a single feature independently from the other system characteristics, and (2) to detect particular combinations of features that may give particularly good performances. The evident problem is that, in the analysis, all the data flows should be studied on all the machines, possibly requiring a prohibitive time.

Turning to a different side, it is evident that this kind of qualitative analysis is strongly biased by the performances and the NN paradigm chosen as a reference. This asks for an accurate choice of the performances, which could easily incorporate new models. In other situations, it may sometimes present definition problems. As a practical example, consider the case of large, sparsely connected networks. The present approach completely ignores the need to deal efficiently with this kind of networks. Using the approach seen so far, a *Sparse Matrix Performance* could be defined as

$$\xi_S = \frac{\text{Throughput on the Sparse Matrix Problem}}{k_S \cdot \text{Normal Throughput}}.$$

The *sparse matrix problem* could still be a matrix-vector multiplication but with only one weight out of k_S being non-zero. The ideal result would be one, that is a throughput increased of a factor k_S, being $1/k_S$ the reduction in arithmetic complexity of the problem. Unfortunately, such measure would return $1/k_S$ in the worst case, whereas a null would be desirable. (No reasonable system could see its own throughput reduced when dealing with a sparse matrix.) Moreover, the application of such a measure is relatively complex because

- the choice of the position of the non-null weights in the matrix should be standardized but its location is likely to affect the measurement in many architectures; and
- the sparse-matrix problem is seldom addressed in the original papers of systems not especially conceived for the purpose.

The last bullet hits another problem of the approach: Architecture descriptions in literature are often rather elusive on mapping problems, especially in non-standard cases as virtualization or efficient use of large hardware for smaller networks. Sometimes, only specific examples of algorithm mapping are discussed and, needless to say, almost always they are optimal cases. This need for an exact knowledge of the data-flow and timing of system must be stressed as one of the most important limits of the approach. Some more complex architectures—especially if based on DSPs or μPs, as the Music system (see §2.2)—could, in principle, be examined under the given framework, but, in practice, may prove either just too complex or requiring too many simplifications and assumptions to fit the proposed scheme. Future work should assess this problem.

The problem encountered in dealing with non-trivial algorithm mapping can also be read as a positive results of this approach: The analysis of optimal mapping of matrix algorithm on generic size monodimensional arrays is a fundamental component of the performance of an architecture, not less important than the physical connectivity of the processing elements. Once more, it can be emphasized that, even if optimal algorithmic mapping on parallel machines is an extremely complex task, neural networks require a small subset of regular operations to be mapped. Further development in the field can be foreseen and useful results likely to be obtained.

Another interesting aspect, not considered before but connected with the documentation difficulties discussed above, is the possibility of using the experience gained in this analytical approach to develop benchmark specifications applicable to existing machines. Some new problems would arise (as the fact of the optimal size $m \times n$ being, at least partly, fixed by the implementation), but the results of the present approach may well help in the task.

4 Conclusions

In this document, several architectures for digital electronic hardware conceived for neural networks computation have been reviewed. Some comments have been made on advantages and limitations of each of the architectures while reviewing the designs. In general, some difficulties have been remarked in performing objective comparisons of the architectural classes examined. To allow a more precise analysis, some quantitative performance measurements have been introduced. Unfortunately the task of a quantitative analysis has shown to be harder than expected, often requiring a more detailed knowledge of the systems than available. Nevertheless, the approach, still rather immature, allows to draw some interesting considerations on the subject of architectural design of neuro-computers.

A first conclusion is that the ability to map larger networks on comparatively small machines is an essential feature of any general-purpose neuro-computer. This statement emphasizes the importance of the optimal mapping of the required computations on the available hardware. The mapping is also critical in the opposite direction, when small networks need to take the best advantage of larger machines. Often researchers in the field seem to focalise on machines conceived—implicitly or explicitly—on one specific (eventually maximal) size of problem. Performances on generic networks critically depend on the mapping algorithms which should be considered together with hardware connectivity details in architectural evaluation.

2D architectures are implicitly less reconfigurable as the data flow is quite rigid. At the same time they have the important advantage of allowing throughputs much higher than 1D architectures. On the other side, 1D architectures should be in principle easier to reconfigure allowing a more efficient mapping of generic-size algorithms. An interesting evolution could be to apply the type of hierarchical design described here for the Systolic Ring of 2D Matrix Processors to a 1D structure improved with respect to the SRA. This would also make possible to exploit the inherent modularity of 1D structures. By *modularity*, it is not meant here the feature expressed in the introduction (see §1.2) but rather the possibility offered to the user to adapt the system to his/her problem complexity or to the available funding. 2D machines may well be modular at the designer level (as MANTRA I is), allowing the designer to reuse a previous design to change the system size; but an evident topological advantage make 1D systems reconfigurable simply by inserting more elements in the linear array with no modification to the external hardware.

References

[ALI92] Cesare Alippi and Marley Vellasco. GALATEA neural VLSI architectures: Communication and control consideartions. *Microprocessing and Microprogramming*, 35:175–80, 1992.

[ASA90] Krste Asanović and Nelson Morgan. Experimental determination of precision requirements for back-propagation training of artificial neural networks. In *1st International International Conference on Microelectronics for Neural Networks*, pages 9–15, Dortmund, Germ., 1990.

[BAK91] Thomas Baker. Artificial neural network and image processing using the Adaptive Solutions' architecture. Paper distributed by Adaptive Solutions, March 1991.

[BEI91A] J. Beichter, N. Brüls, E. Meister, U. Ramacher, and H. Klar. Design of a general-purpose neural signal processor. In *2nd International Conference on Microelectronics for Neural Networks*, pages 311–15, 1991.

[BEI91B] J. Beichter, U. Ramacher, and H. Klar. VLSI design of a neural signal processor. In Mariagiovanna Sami and Jesus Calzadilla-Daguerre, editors, *Silicon Architectures for Neural Nets*, pages 245–60. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1991.

[CHU92] Jai-Hoon Chung, Hyunsoo Yoon, and Seoul Ryoul Maeng. A systolic array exploiting the inherent parallelisms of artificial neural networks. *Microprocessing and Microprogramming*, 33:145–59, 1992.

[DAR88] DARPA. *Neural Network Study.* AFCEA International Press, Fairfax, Va., 1988.

[DEG89] A. J. De Groot and S. R. Parker. Systolic implementation of neural networks. In *SPIE High Speed Computing II*, volume 1058, pages 182–90, Los Angeles, Calif., 1989.

[FRA92] Edoardo Franzi. Accélérateur par circuit neuronal L-Neuro 1.0. CPIT Project (École Polytechnique Fédérale de Lausanne), December 1992.

[FUL77] Samuel H. Fuller and William E. Burr. Measurement and evaluation of alternative computer architectures. *Computer*, 10(10):24–35, October 1977.

[GRI91] Matthew Griffin, Gary Tahara, Kurt Knorpp, and Bob Riley. An 11-million transistor neural network execution engine. In *IEEE International Conference on Solid-State Circuits*, pages 180–81, 1991.

[GUE92] Anne Guérin-Dugue. Machines pour réseaux de neurones artificiels. In *Cours Post-grade en Informatique Technique: Réseaux de neurones biologiques et artificiels.* École Polytecnique Fédérale de Lausanne, Lausanne, September 1992. Lecture Notes.

[HAM] Dan Hammerstrom and Nguyen Nguyen. An implementation of Kohonen's self-organizing map on the Adaptive Solutions neurocomputer. Paper distributed by Adaptive Solutions.

[HAM91] Dan Hammerstrom. A highly parallel digital architecture for neural network emulation. In José G. Delgado-Frias and William R. Moore, editors, *VLSI for Artificial Intelligence and Neural Networks*, chapter 5.1, pages 357–66. Plenum Press, New York, 1991.

[HAY92] John Haynes, October 1992. Personal Communication.

[HEN90] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, San Mateo, Calif., 1990.

[HIR92] Yuzo Hirai. *VLSI Neural Network Systems*. Japanese Technology Reviews. Gordon and Breach Science Publishers, Switzerland, 1992.

[HNC93] HNC Inc., Santa Diego, Calif. *HNC100 SIMD Neurocomputer Array Processor (SNAP) Chip and System Specification*, 1993. Preliminary Data.

[HUN91] Jamie Hunter and Simon Jones. The design of controllers for linear arrays. In *2^{nd} International Conference on Microelectronics for Neural Networks*, pages 197–205, 1991.

[IEN92] Paolo Ienne and Marc A. Viredaz. The GENES Auxiliary Circuit for the Delta rule. Internal Report R92.19, École Polytecnique Fédérale de Lausanne - LAMI, October 1992.

[INO] inova microelectronics corporation, Santa Clara, Calif. *N64000 Digital Neural Network Processor*. Preliminary Data.

[JON91A] S. Jones, K. Sammut, Ch. Nielsen, and J. Staunstrup. *Toroidal Neural Network: Architecture and Processor Granularity Issues*, pages 229–254. Kluwer Academic Publishers, Norwell, Mass., 1991.

[JON91B] Simon Jones, Karl Sammut, and Jamie Hunter. Toroidal neural network processor: Architecture, operation, performance. In *2^{nd} International Conference on Microelectronics for Neural Networks*, pages 163–69, 1991.

[KUN90] S. Y. Kung and John Vlontzos. Special purpose array processor implementation of neural networks. In *IEEE 2^{nd} Conference on Tools for Artificial Intelligence*, pages 130–37, Herndon, Va., 1990.

[LEH91] 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 325–34. Plenum Press, New York, 1991.

[MAU92] Nicolas Mauduit, Marc Duranton, Jean Gobert, and Jacques-Ariel Sirat. Lneuro 1.0: A piece of hardware LEGO for building neural network systems. *IEEE Transactions on Neural Networks*, 3(3):414–22, May 1992.

[MCC91] Hal McCartor. Back propagation implementation on the Adaptive Solutions neurocomputer chip. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 3. Morgan Kaufmann Publishers, San Mateo, Calif., 1991.

[MEA91] Robert W. Means and Layne Lisenbee. Extensible linear floating point SIMD neurocomputer array processor. In *International Joint Conference on Neural Networks*, volume I, pages 587–92, Seattle, Wash., July 1991.

[MEA93] Robert W. Means and Layne Lisenbee. Floating-point SIMD neurocomputer array processor. Paper distributed by HNC Inc., 1993.

[MOR92] Nelson Morgan, James Beck, Phil Kohn, Jeff Bilmes, Eric Allman, and Joachim Beer. The ring array processor: A multiprocessing peripheral for connectionist applications. *Journal of Parallel and Distributed Computing*, 14:248–59, 1992.

[MUL92] Urs A. Müller, Bernhard Bäumle, Peter Kohler, Anton Gunzinger, and Walter Guggenbül. Achieving supercomputer performance for neural net simulation with an array of digital signal processors. *IEEE Micro*, pages 55–65, October 1992.

[NIE91] C.D. Nielsen, J. Staunstrup, and J. R. Jones. A delay-insensitive neural network engine. In José G. Delgado-Frias and William R. Moore, editors, *VLSI for Artificial Intelligence and Neural Networks*, chapter 5.2, pages 367–76. Plenum Press, New York, 1991.

[RAM89] U. Ramacher and J. Beichter. Systolic architectures for fast emulation of artificial neural networks. In John McCanny, John McWhirter, and Earl Swartzlander, Jr., editors, *Systolic Array Processors*, pages 277–86. Prentice Hall, New York, 1989.

[RAM90] Ulrich Ramacher and Wolfgang Raab. Fine-grain architectures for systolic emulation of neural algorithms. In *IEEE International Conference on Application Specific Array Processors*, pages 554–66, 1990.

[RAM91A] U. Ramacher, J. Beichter, and N. Brüls. Architecture of a general-purpose neural signal processor. In *International Joint Conference on Neural Networks*, volume I, pages 443–46, Seattle, Wash., July 1991.

[RAM91B] U. Ramacher, J. Beichter, W. Raab, J. Anlauf, N. Brüls, U. Hachmann, and M. Wesseling. *Design of a* 1^{st} *Generation Neurocomputer*, pages 271–310. Kluwer Academic Publishers, Norwell, Mass., 1991.

[RAM91C] U. Ramacher and P. Nachbar. Hamiltonian dynamics of neural networks. In *2^{nd} International Conference on Microelectronics for Neural Networks*, pages 95–102, 1991.

[RAM91D] U. Ramacher, W. Raab, J. Anlauf, U. Hachmann, and M. Wesseling. *SYNAPSE-X*: A general-purpose neurocomputer. In *2^{nd} International Conference on Microelectronics for Neural Networks*, pages 401–9, 1991.

[SEJ87] Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–68, 1987.

[THE90] J. B. Theeten, M. Duranton, N. Mauduit, and J. A. Sirat. The Lneuro chip: A digital VLSI with on-chip learning mechanism. In *International Neural Networks Conference*, pages 593–96, 1990.

[VIR92] 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, September 1992.

[VLO91] J. A. Vlontzos and S. Y. Kung. *Digital Neural Network Architecture and Implementation*, pages 205–227. Kluwer Academic Publishers, Norwell, Mass., 1991.

[WEI93] Andreas S. Weigend and Neil A. Gershenfeld. Results of the Time Series Competition at the Santa Fe Institute. In *IEEE International Conference on Neural Networks*, San Francisco, Calif., 1993. Submitted Paper.

[YAS90] Moritoshi Yasunaga, Noboru Masuda, Masayoshi Yagyu, Mitsuo Asai, Minoru Yamada, and Akira Masaki. Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed of 576 digital neurons. In *International Joint Conference on Neural Networks*, volume II, pages 527–36, 1990.

[YAS91] Moritoshi Yasunaga, Noboru Masuda, Masayoshi Yagyu, Mitsuo Asai, Katsunari Shibata, Mitsuo Ooyama, Minoru Yamada, Takahiro Sakaguchi, and Masashi Hashimoto. A self-learning neural network composed of 1152 digital neurons in wafer-scale LSIs. In *International Joint Conference on Neural Networks*, pages 1844–49, Seattle, Wash., July 1991.

A Evaluation of the Quantitative Criteria

This appendix sketches the calculations used to evaluate the quantitative criteria described in §3.3 and whose results are shown in the tables therein contained. Each of the following subsections is devoted to the calculation for one architecture (or one architecture under a given set of assumptions).

A.1 Broadcast Bus Architecture

The BBA is strongly penalized if the weights have to be swapped during the calculation. Apart from the basic case (minimum storage to maximize forward throughput), it is fair enough to consider also the case of a machine with sufficient storage for the transpose weights.

A.1.1 Type I: No Allowance for Efficient Transposition

The parameters are computed as follows:

1. **Number of Processors** (N_P). The physical array has one processor per neuron, thus $N_P = m$.

2. **Weight and Data inputs** (I_W and I_x). It is characteristic of this architecture the single input 8-bit bus. Not only $I_W = I_x = 1$ for any value of m and n, but the input is also shared for the weights or data.

3. **Result outputs** (O_p). Similarly $O_p = 1$, that is, there is also a single output 8-bit bus.

4. **Data Storage per Processor** (S_P). Each processor (neuron) contains the weights for his synapses. Therefore it contains n weights one register for the result accumulation between cycles. If n is large, $S_P \simeq n$.

5. **Epoch Size** (E). A single vector is computed at a time, $E = 1$.

6. **Latency** (λ). Each processor adds to the accumulator a synapse at a time, thus $\lambda = n$.

7. **Throughput** (θ). As it is not a pipelined architecture, $\theta = 1/\lambda = 1/n$. That is, one vector is output every n cycles.

8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = nm \cdot \frac{1/n}{m} = 1$, that is all the processors are active all the time.

9. **Transposition Performance** (ξ_T). The throughput on the transpose matrix is $\theta_T = \frac{1}{m\lceil n/m \rceil + 2mn}$. The first term in the denominator corresponds to the number of cycles needed to compute the output and the second accounts for the need to swap the matrix (mn terms exchanged one per cycle, once to send the transpose and once more to resend the original matrix). Consequently, $\xi_T = \frac{n}{m\lceil n/m \rceil + 2mn}$. Clearly the term $2mn$ dominates the denominator, meaning that the performance loss due to the weight swap is enormous.

10. **Reconfiguration Performance** (ξ_R). Neurons are processed sequentially on the processors in k_R groups of m. Each group requires n/k_R accumulations, giving the same throughput as in the basic case. Thus, $\xi_R = 1$.

11. **Virtualization Performance** (ξ_V). The throughput of the system on a $k_V m \times k_V n$ matrix is $\theta_V = 1/(k_V(k_V n) + k_V^2 mn)$, where $k_V n$ is the number of cycles to compute m neuron outputs in parallel. This is repeated k_V times to compute all the $k_V m$ neuron outputs. The term $k^2 mn$ accounts for the needed swapping of the k_V^2 $k_V m \times k_V n$ submatrixes. This results in $\xi_V = 1/(m+1)$, clearly extremely penalized by the presence of m at the denominator.

A.1.2 Type II: Dimensioned for Efficient Transposition and with Epoch Processing

The parameters are computed as follows:

1. **Number of Processors** (N_P). The physical array has, at least, one processor per matrix row in either orientation, thus $N_P = \max(m, n)$.
2. **Weight and Data inputs** (I_W and I_x). No change. $I_W = I_x = 1$.
3. **Result outputs** (O_p). No change. $O_p = 1$.
4. **Data Storage per Processor** (S_P). Each typical processor contains the elements of a row of the weight matrix plus those of a column. Some has only columns or only rows depending on the ratio m/n. If all processors are equal, again neglecting the accumulator, $S_P \simeq m + n$.
5. **Epoch Size** (E). An epoch length, intended as a number of vectors computed between every matrix swap, can be fixed arbitrarily to $E = E^\star$.
6. **Latency** (λ). No change. $\lambda = n$.
7. **Throughput** (θ). No change. $\theta = 1/\lambda = 1/n$.
8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = nm \cdot \frac{1/n}{\max(m,n)} = \frac{m}{\max(m,n)}$.
9. **Transposition Performance** (ξ_T). The throughput on the transpose matrix is $\theta_T = \frac{1}{m}$. Thus, $\xi_T = \frac{n}{m}$. This indicates that the parallelization efficiency is one either for the direct or for the transpose problem, the other having an efficiency $\frac{\min(m,n)}{\max(m,n)}$.
10. **Reconfiguration Performance** (ξ_R). Neurons are processed sequentially on the processors in $k_R m/\max(m, n)$ groups of $\max(m, n)$. Each group requires n/k_R accumulations, giving $\xi_R = \max(m, n)/m$.
11. **Virtualization Performance** (ξ_V). The throughput of the system on a $k_V m \times k_V n$ matrix and with an epoch length $E^\star$ is $\theta_V = E^\star/(k_V(k_V n)E^\star + k_V^2 mn)$. Again $k_V n$ is the number of cycles to compute m neuron outputs in parallel. This is repeated k_V times to compute all the $k_V m$ neuron outputs and for $E^\star$ neurons. The term $k_V^2 mn$ accounts for the needed swapping of the k_V^2 $k_V m \times k_V n$ submatrixes. This results in $\xi_V = E^\star/(m + E^\star)$, becoming one for $E^\star \gg m$.

A.2 Systolic Ring Architecture

The processor allocation choices have been briefly discussed in §3.2.2. SRA types MIN and MAX have consistent differences, deserving separate calculations.

A.2.1 Type MIN: Composed of $\min(m, n)$ Processors

The parameters are computed as follows:

1. **Number of Processors** (N_P). The physical array is composed of $N_P = \min(m, n)$ processors. The ring together with the FIFO should contain $\max(m, n)$ elements, either for the input values to be circulated to the processors, either for the output accumulators. This requires a FIFO of $\max(m, n) - \min(m, n)$ words.

2. **Weight and Data inputs** (I_W and I_x). Each processor has a weight input at the top. Thus, $I_W = N_P = \min(m, n)$. All the processors and all the memory elements in the FIFO—if $n = \max(m, n)$—are accessible in parallel to load them with the input vector. If $n = \min(m, n)$, only the processors are accessible to store the input. In any case, $I_x = n$.

3. **Result outputs** (O_p). Each processor either locally accumulates an output word during the multiplication process or the accumulation is done on the ring. In both cases, $O_p = m$.

4. **Data Storage per Processor** (S_P). Each processor has simply to accumulate the result of the multiplication of the weights and the inputs. Roughly, this needs about $S_P \simeq 2 \sim 3$ bits per weight bit.

5. **Epoch Size** (E). The computations are performed on a single vector at a time, thus $E = 1$.

6. **Latency** (λ). The latency is given by the number of multiply and accumulate operations necessary to accumulate the first value, $\lambda = n$.

7. **Throughput** (θ). The number of multiply and accumulate operations to perform is mn on $\min(m, n)$ processors. Then, $\theta = 1/\max(m, n)$.

8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = nm \cdot \frac{1/\max(m,n)}{\min(m,n)} = 1$.

9. **Transposition Performance** (ξ_T). The same number of operations have to be performed with the same number of processors. This implies $\xi_T = 1$.

10. **Reconfiguration Performance** (ξ_R). If $m < n$ each processor contains all the data for a neuron. Thus the ring has to be reused k_R times to compute all the potentials. The length of each cycle throughout the ring is n/k_R, but no less than the number of processors (m). Hence, $\theta_R = \frac{1/k_R}{\max(m,n/k_R)}$. If $m \geq n$ instead, the exceeding processors—dedicated to matrix columns no longer existing in the rearranged matrix—cannot be reused under the data flow description hypothesized. Now, the length of each cycle throughout the ring is $k_R \cdot m$, but no less than the number of processors (n); hence $\theta_R = 1/(k_R m)$ In general, $\theta_R = \frac{1/k_R}{\max(m,n/k_R)}$ and $\xi_R = \frac{\max(m,n)}{k_R \max(m,n/k_R)}$.

11. **Virtualization Performance** (ξ_V). The ring is reused (possibly over the same accumulation line, if $m > n$) k_V times, each cycle being k_V times longer than in the basic problem. This gives $\xi_V = 1$.

A.2.2 Type MAX: Composed of $\max(m, n)$ Processors

The parameters are computed as follows:

1. **Number of Processors** (N_P). The physical array is composed of $N_P = \max(m, n)$ processors.

2. **Weight and Data inputs** (I_W and I_x). Each processor has a weight input at the top. Thus, $I_W = N_P = \max(m, n)$. One data path is provided for each input element. Hence, $I_x = n$.

3. **Result outputs** (O_p). One data path is provided for each output element. Hence, $O_p = m$.

4. **Data Storage per Processor** (S_P). Each processor has simply to accumulate the result of the multiplication of the weights and the inputs. Roughly, this needs about $S_P \simeq 2 \sim 3$ bits per weight bit.

5. **Epoch Size** (E). The computations are performed on a single vector at a time, thus $E = 1$.

6. **Latency** (λ). The latency is given by the number of multiply and accumulate operations necessary to accumulate the first value, $\lambda = n$.

7. **Throughput** (θ). The available $\max(m, n)$ processors allow to perform—more or less efficiently—a full matrix-vector product every n cycles. Then, $\theta = 1/n$.

8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = \frac{m}{\max(m,n)}$.

9. **Transposition Performance** (ξ_T). As in the case of the direct matrix, $\max(m, n)$ processors always allow to perform a full transpose matrix-vector product every m cycles. This implies $\xi_T = n/m$.

10. **Reconfiguration Performance** (ξ_R). Different arrangements of the weights should be considered considering whether $k_R m > n/k_R$ or $k_R m < n/k_R$. To simplify, $k_R > \sqrt{\frac{n}{m}}$ can be assumed. Considering that the case $k_R = \frac{n}{m}$ corresponds to a matrix of the same size of the transpose, it is reasonable to suppose that the interest is here in matrixes more *deformed* than the transpose. Thus, when $k_R m > n/k_R$, the neurons are processed in $\lceil k_R m / \max(m, n) \rceil$ groups of $\max(m, n)$, each group requiring n/k_R cycles to produce the result. Hence, $\xi_R = k_R / \left\lceil \frac{k_R m}{\max(m,n)} \right\rceil$.

11. **Virtualization Performance** (ξ_V). If $m \geq n$, the ring is reused k_V times, each cycle being k_V times longer than in the basic problem. This gives $\xi_V = 1$. If $m < n$, the accumulation is performed on the ring. $\left\lceil \frac{k_V n}{\max(m,n)} \right\rceil$ groups are considered, each requiring $\left\lceil \frac{k_V m}{\max(m,n)} \right\rceil$ complete rotations of the ring, which in turn require $\max(m, n)$ cycles. Being $n = \max(m, n)$, ξ_V is easily obtained. It can be written as $\xi_V = k_V / \left\lceil \frac{k_V \min(m,n)}{n} \right\rceil$, valid for any m/n ratio.

A.3 Systolic Ring Architecture with Global Bus

The idea of the global bus, added to a conventional SRA, is to minimize the inefficiencies of SRA/MAX. The parameters often show similarities with the SRA/MAX:

1. **Number of Processors** (N_P). The physical array is composed of $N_P = \max(m, n)$ processors.

2. **Weight and Data inputs** (I_W and I_x). Each processor has a weight input at the top. Thus, $I_W = N_P = \max(m,n)$. One data path is provided for each input element. Hence, $I_x = n$.

3. **Result outputs** (O_p). One data path is provided for each output element. Hence, $O_p = m$.

4. **Data Storage per Processor** (S_P). Each processor has simply to accumulate the result of the multiplication of the weights and the inputs. Roughly, this needs about $S_P \simeq 2 \sim 3$ bits per weight bit.

5. **Epoch Size** (E). The computations are performed on a single vector at a time, thus $E = 1$.

6. **Latency** (λ). If $m > n$, the computation is performed exactly as with the SRA/MAX, giving a latency of n cycles. Otherwise—whereas part of the processors are kept idle in the SRA/MAX to directly add all the terms of a result—here all the units are kept busy during m cycles and $\lceil n/m \rceil$ partial terms per result component are produced with maximum efficiency. These are then added using the global bus. It takes $(\lceil n/m \rceil - 1)$ cycles to sum the first output element. In general, $\lambda = \min(m,n) + \left\lceil \frac{n}{\min(m,n)} - 1 \right\rceil$.

7. **Throughput** (θ). The same procedure used to calculate the latency can be followed. It should be noted that adding all the partial terms with the global bus takes a total of $(\lceil n/m \rceil - 1) \cdot m$ cycles. Hence, it results $\theta = \frac{1}{\lceil \max(m,n)/m \rceil \min(m,n)}$.

8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = \frac{1}{\lceil \max(m,n)/m \rceil}$.

9. **Transposition Performance** (ξ_T). The situation is perfectly dual to that described for the throughput. This implies $\theta_T = \frac{1}{\lceil \max(m,n)/n \rceil \min(m,n)}$, and $\xi_T = \frac{\lceil \max(m,n)/m \rceil}{\lceil \max(m,n)/n \rceil}$.

10. **Reconfiguration Performance** (ξ_R). Different arrangements of the weights should be considered considering whether $k_R m > n/k_R$ or $k_R m < n/k_R$. If it is assumed $k_R > \sqrt{\frac{n}{m}}$ as done in A.2.2, it follows $k_R m > n/k_R$ and hence the neurons are processed in $\lceil k_R m / \max(m,n) \rceil$ groups of $\max(m,n)$, each group requiring n/k_R cycles to produce the result. This is the same result obtained in the case of SRA/MAX. Hence, $\xi_R = \frac{k_R \frac{m}{\max(m,n)} \left\lceil \frac{\max(m,n)}{m} \right\rceil}{\left\lceil \frac{k_R m}{\max(m,n)} \right\rceil}$.

11. **Virtualization Performance** (ξ_V). If $m \geq n$, the ring is reused k_V times, each cycle being k_V times longer than in the basic problem. This gives $\xi_V = 1$. If $m < n$, the accumulation is performed on the ring over $\lceil \max(m,n)/(k_V m) \rceil$ partial terms per result component. $\left\lceil \frac{k_V n}{\max(m,n)} \right\rceil$ groups are considered, each requiring $\lceil k_V m \rceil$ cycles. If k_V is sufficiently large ($k_V > n/m$), only one partial term is generated in all groups except the last, which can be neglected for simplicity. This implies that global bus is not used (except in the last group) and, still for $m < n$, the throughput is $\theta_V = 1/(k_V^2 m)$. It results $\xi_V = \left\lceil \frac{n}{\min(m,n)} \right\rceil$, valid for any m/n ratio.

A.4 Multiple Systolic Ring Architecture

The number of rings is fixed to N. The results are often similar to the SRA/MIN (see A.2):

1. **Number of Processors** (N_P). The same considerations made on the SRA apply. Now $N_P = N \cdot \min(m, n)$ processors are needed to replicate N times the ring.
2. **Weight and Data inputs** (I_W and I_x). Each processor has a weight input at the top and this is shared by the N rings. Conversely, separate input paths are provided on each ring. Thus, $I_W = \min(m, n)$ and $I_x = N \cdot n$.
3. **Result outputs** (O_p). Each processor in each ring locally accumulates an output word during the multiplication process, thus $O_p = N \cdot m$.
4. **Data Storage per Processor** (S_P). As with the SRA, $S_P \simeq 2 \sim 3$ bits per weight bit.
5. **Epoch Size** (E). To use efficiently each ring, $E = N$ vectors should be injected at the same time.
6. **Latency** (λ). As with the SRA, $\lambda = n$.
7. **Throughput** (θ). N new vectors can be presented every $\max(m, n)$ cycles. Hence, $\theta = N/\max(m, n)$.
8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = 1$.
9. **Transposition Performance** (ξ_T). As with the SRA, the same number of operations have to be performed with the same number of processors. This implies $\xi_T = 1$.
10. **Reconfiguration Performance** (ξ_R). As with the SRA, $\xi_R = \frac{\max(m,n)}{k_R \max(m,n/k_R)}$.
11. **Virtualization Performance** (ξ_V). As with the SRA, $\xi_V = 1$.

A.5 Systolic Square Array Architecture

The parameters are computed as follows:

1. **Number of Processors** (N_P). The physical array has to be a square, thus the square side should contain $\max(m, n)$ elements. Thus, $N_P = \max^2(m, n)$.
2. **Weight and Data inputs** (I_W and I_x). Each row of processors has a weight input at the top. Each processor on the top and left side of a square has a data input, one side at time being used for the input vector (in the forward phase considered here). Only n of these $\max(m, n)$ are used on the regular size matrix, thus $I_W = I_x = n$.
3. **Result outputs** (O_p). Similarly each processor on the bottom and right side has an output, one side at time being used for the result. Again, only m of these $\max(m, n)$ are used, thus $O_p = m$.
4. **Data Storage per Processor** (S_P). In the case of GENES IV, each processor contains four 40-bit data registers and two 32-bit weight registers. This is the minimum to support the data circulation defined for the SSAA, including the delays to diagonalize the data input. Therefore, $S_P = 7$.

5. **Epoch Size** (E). The epoch size depends (in a practical situation) also on the delay added by the external hardware in series on the data-path (e.g., the non-linear function). In the simplest case (only matrix-vector multiplication performed) the pipeline depth depends only on the array size and $E = 2\max(m,n)$. For large values of m and n, the importance of the external hardware becomes negligible.

6. **Latency** (λ). The latency is the element that determines the epoch length and consequently it is simply $\lambda = 2\max(m,n)$.

7. **Throughput** (θ). One vector can be presented to the inputs every 40 clock cycles and one vector is output at the same rate. Forty clock cycles are also indicated as a macro-cycle and correspond to a multiply/add operation. Then, $\theta = 1$.

8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = \frac{nm}{\max^2(m,n)} = \frac{\min(m,n)}{\max(m,n)}$.

9. **Transposition Performance** (ξ_T). The array being a square, the transposition has no effect on the throughput. This implies $\xi_T = 1$.

10. **Reconfiguration Performance** (ξ_R). The weight exchange during the rearranged problem calculation is completely transparent in the sense that it is performed in the background while the previous computation is performed. Thus the throughput is simply the inverse of the number of square $\max(m,n) \times \max(m,n)$ submatrixes contained in the $k_R m \times n/k_R$ matrix. It results $\xi_R = 1/\left(\left\lceil\frac{k_R m}{\max(m,n)}\right\rceil \left\lceil\frac{n/k_R}{\max(m,n)}\right\rceil\right)$.

11. **Virtualization Performance** (ξ_V). The weight exchange during a virtual size problem computation is completely transparent, in the sense that it is performed in the background while the previous computation is being completed. Thus the throughput is simply the inverse of the number of square $\max(m,n) \times \max(m,n)$ submatrixes contained in the $k_V m \times k_V n$ matrix. After few elementary passages, $\xi_V = k_V/\left\lceil k_V \cdot \frac{\min(m,n)}{\max(m,n)}\right\rceil$. Being $\frac{\min(m,n)}{\max(m,n)} \leq 1$, it follows $\xi_V \geq 1$ for any $k_V \in \mathbb{N}$, non-zero.

A.6 Systolic Ring of Matrix Processors Architecture

The computations are the same for Type I and Type II. The results are in some cases functions of the epoch size of the matrix processor (E_{MP}), of its latency (λ_{MP}) and its throughput (θ_{MP}). Using matrix processors for square matrixes of size N, the following applies:

	E_{MP}	λ_{MP}	θ_{MP}
Type I	N	N^2	$1/N$
Type II	$2N$	$2N$	1

The values given for Type I are derived from the SIEMENS Elementary Chain structure, those for Type II are the values computed for a normal SSAA. Problems and results are often similar to the SRA/MIN (see A.2).

1. **Number of Processors** (N_P). The same considerations made on the SRA/MIN applies. The ring contains $\lceil\min(m,n)/N\rceil$ matrix processors, each containing N or N^2 scalar processors—for Type I and II respectively. Thus $N_P = \lceil\min(m,n)/N\rceil \cdot N$ for Type I and $N_P = \lceil\min(m,n)/N\rceil \cdot N^2$ for Type II.

2. **Weight and Data inputs** (I_W and I_x). In Type I, each matrix processor has a weight input at the top, thus obtaining $I_W = \lceil \min(m,n)/N \rceil$. Type II matrix processors have N inputs each, totalling $I_W = \min(m,n)$. Similarly for the data inputs, $I_x = \lceil n/N \rceil$ and $I_x = n$ respectively[8].

3. **Result outputs** (O_p). The discussion of the data inputs applies also to the data outputs, then $O_p = \lceil m/N \rceil$ and $O_p = m$ respectively[9].

4. **Data Storage per Processor** (S_P). The SIEMENS Elementary Chains hold two local copies of the three matrixes (input data, weights, accumulator), totalling an $S_P = 6$. The GENES IV style processor has $S_P = 7$.

5. **Epoch Size** (E). The ring processing a single matrix input at a time, $E = E_{MP}$.

6. **Latency** (λ). The latency is given by the number of multiply and accumulate operations performed by each matrix processor times the latency of each matrix processor, $\lambda = \lceil n/N \rceil \cdot \lambda_{MP}$.

7. **Throughput** (θ) The system throughput is the matrix processor throughput divided by the number of cycles required by the ring to produce the output. Then, similarly to the SRA, $\theta = \frac{1}{\lceil \max(m,n)/N \rceil} \cdot \theta_{MP}$.

8. **Parallelization Efficiency** (η_Q). From the definition, $\eta_Q = \frac{nm}{\lceil \max(m,n)/N \rceil \lceil \min(m,n)/N \rceil N^2} = \frac{nm}{\lceil n/N \rceil \lceil m/N \rceil N^2}$ in both cases. This indicates that all the processors are active all the time—either in the direct multiplication or in the transpose—with the sole exceptions due to the divisibility of n and m by N.

9. **Transposition Performance** (ξ_T). As with the SRA, the same ring length should be considered for the transpose; hence $\xi_T = 1$, independently of the θ_{MP} value.

10. **Reconfiguration Performance** (ξ_R). If $m < n$ each processor contains all the data for N neurons. Thus the ring has to be reused k_R times to compute all the potentials. The length of each cycle throughout the matrix processor ring is $\left\lceil \frac{n/k_R}{N} \right\rceil$, but no less than the number of processors ($\lceil m/N \rceil$). Hence, $\theta_R = \frac{1/k_R}{\max(\lceil m/N \rceil, \left\lceil \frac{n/k_R}{N} \right\rceil)} \cdot \theta_{MP}$. If $m \geq n$ instead, the exceeding processors—dedicated to matrix columns no longer existing in the rearranged matrix—cannot be reused under the data flow description hypothesized. Now, the length of each cycle throughout the ring is $\left\lceil \frac{k_R \cdot m}{N} \right\rceil$, but no less than the number of processors ($\lceil n/N \rceil$); hence $\theta_R = \frac{1}{\left\lceil \frac{k_R \cdot m}{N} \right\rceil} \cdot \theta_{MP}$ In general, assuming $k_R \cdot \lceil \alpha \rceil \simeq \lceil k_R \cdot \alpha \rceil$, $\xi_R = \frac{\lceil \max(m,n)/N \rceil}{k_R \lceil \max(m, n/k_R)/N \rceil}$.

11. **Virtualization Performance** (ξ_V). The ring is reused (possibly over the same accumulation line, if $m > n$) k_V times, each cycle being approximately k_V times longer than in the basic problem. More precisely, each cycle is long $\left\lceil \frac{k_V \cdot \max(m,n)}{N} \right\rceil$. This gives $\xi_V = \frac{k_V \lceil \max(m,n)/N \rceil}{\lceil k_V \cdot \max(m,n)/N \rceil}$. This is one for m and n integer multiple of N.

[8] This does not exactly describe the SIEMENS architecture as in that case the inputs are not parallel. This only requires some up- and down-loading overhead. The present description is consistent with the SRA analysis.

[9] See note 8.

B Back-propagation Algorithm

This appendix contains the definition of the back-propagation algorithm in the form used in the present document. It does not takes into account specific variations of the algorithm. It is rather intended as a notation reference for the rest of the document.

The described network has L layers. The reference layer q is shown in square brackets at the exponent. Thus, for instance, $\mathbf{W}^{[q]}$ is the weight matrix connecting the output of the $(L-1)$-th layer to the input of the L-th. Each layer has M_q neurons.

The outputs $\vec{\mathbf{y}}$ of each layer are computed as the application of a non-linear function $\sigma(\bullet)$ to the potential $\vec{\mathbf{p}}$, as follows:

$$
\begin{aligned}
\vec{\mathbf{y}}^{[1]} &= \sigma(\vec{\mathbf{p}}^{[1]}) \\
&= \sigma(\mathbf{W}^{[1]} \cdot \vec{\mathbf{x}}) \qquad (1) \\
\vec{\mathbf{y}}^{[q]} &= \sigma(\vec{\mathbf{p}}^{[q]}) \\
&= \sigma(\mathbf{W}^{[q]} \cdot \vec{\mathbf{y}}^{[q-1]}) \qquad \text{for } q = 2, 3, \ldots, L, \qquad (2)
\end{aligned}
$$

where $\mathbf{W}$ is the weight matrix and $\vec{\mathbf{x}}$ the input vector. The error signal δ is computed as:

$$
\begin{aligned}
\delta_i^{[L]} &= \left(d_i - y_i^{[L]}\right) \cdot \sigma'(p_i^{[L]}) \qquad (3) \\
\delta_i^{[q]} &= \left(\sum_{k=1}^{M_{q+1}} W_{k,i}^{q+1} \cdot \delta_k^{q+1}\right) \cdot \sigma'(p_i^{[q]}) \qquad \text{for } q = 1, 2, \ldots, L-1; \qquad (4)
\end{aligned}
$$

where $\sigma'(v) = \mathrm{d}\sigma(v)/\mathrm{d}v$ and $\vec{\mathbf{d}}$ is the vector of the desired outputs. Once the errors has been back-propagated, the weights can be updated:

$$
\begin{aligned}
\mathbf{W}_i^{[1]\,\mathrm{NEW}} &= \mathbf{W}_i^{[1]\,\mathrm{OLD}} + \alpha \cdot \delta_i^{[1]} \cdot \vec{\mathbf{x}}^{\mathrm{T}} \qquad (5) \\
\mathbf{W}_i^{[q]\,\mathrm{NEW}} &= \mathbf{W}_i^{[q]\,\mathrm{OLD}} + \alpha \cdot \delta_i^{[q]} \cdot \vec{\mathbf{y}}^{[q-1]\mathrm{T}} \qquad \text{for } q = 2, 3, \ldots, L; \qquad (6)
\end{aligned}
$$

where the exponents NEW and OLD indicate the weight matrix before and after the update. Equations (3) to (6) are valid for $i = 1, 2, \ldots, M_q$.