A New Concept for Parallel Neurocomputer Architectures

Alfred Strey[1] and Narcís Avellana[2]

[1] Abteilung Neuroinformatik
Universität Ulm, D-89069 Ulm, Germany
[2] Departemento de Diseño de CIs
Universidad Autonoma de Barcelona - C.N.M., 08193 Bellaterra, Spain

Abstract. This paper presents a new concept for a parallel neurocomputer architecture which is based on a configurable neuroprocessor design. The neuroprocessor adapts its internal parallelism dynamically to the required data precision for achieving an optimal utilization of the available hardware resources. This is realized by encoding a variable number of p different data elements in one very long data word of b bits. All components of the neuroproccessor (multiplier, accumulator, adder, ...) support the parallel execution of p operations on all data elements of one very long data word.

1 Introduction

In recent years many neurocomputer architectures have been proposed [3]. Many of them are based on special-purpose processors (so-called *neuroprocessors*) which have been developped for the simulation of neural networks (e.g. [1], [4], [7]). In principle, all such systems can support many neural network models but due to several decisions of the processor design they are optimized only for one or several models. Such decisions may concern

- the precision by which the arithmetical units perform the basic operations,
- the number and arrangement of the arithmetical units placed on one chip,
- the necessity to use pattern parallelism to reduce the number of I/O pins.

The precision is mostly limited to 16 bit fixed point for the weights of a neural network and to 8 bit fixed point for the neuron outputs. In case of the *multilayer perceptron* and the standard *error backpropagation* learning algorithm this precision was shown to be sufficient in most cases [2]. However *self-organizing feature maps* can learn very well with only 6 bit weights [6]. For many other neural network models no analysis about the required precision is available, at least for recurrent neural networks an arithmetical precision of only 16 bit seems not to be sufficient.

The inherent parallelism of neural networks allows the parallel processing of *synapses* and *neurons*. However the parallel processing of *patterns* which is required by several neurocomputers to achieve a high performance can only be used for certain neural network models and also decreases the convergence speed of the learning algorithm.

2 A Configurable Neuroprocessor

The design of the new neuroprocessor described in this paper is based on the following ideas:

1. Instead of reducing the total number of I/O pins by using pattern parallelism the available I/O pins of the neuroprocessor should be utilized optimally for different data precisions by coding p data I/O elements compactly in one I/O word of b bits (see Fig. 1).
2. The maximum precision of all data elements should be 24 bit fixed point.
3. The precision by which the arithmetical operations are performed should be variable and related to the internal degree of parallelism (see Fig. 1).

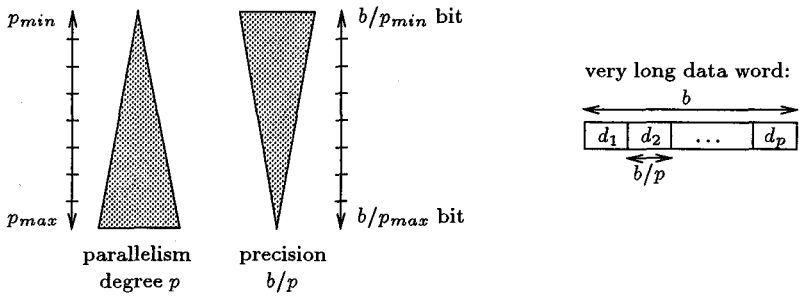

Fig. 1. The concept of hardware configurability

A certain number p of data elements with precision b/p are encoded in one *very long data word* of b bits. So the degree of parallelism p is increased when the required data precision b/p is decreased and vice versa. In order to achieve always a high utilization the hardware resources should be *configured* dynamically to allow parallel operations on all p data elements encoded in one data word.

4. All massively parallel neural operations (related to the simulation of the synapses of a weight matrix W) should be supported efficiently by the neuroprocessor, e.g.:
   - product of a vector $\mathbf{u}$ with the weight matrix W: $x_j = \sum_i u_i \cdot w_{ij}$
   - computation of the squared Euclidean distance between a vector $\mathbf{u}$ and the columns of the matrix W: $d_j = \sum_i (u_i - w_{ij})^2$
   - outer vector product: $\Delta w_{ij} = u_i \cdot \delta_j$

Fig. 2 illustrates the problem of fixed point arithmetics for variable word lengths. In the first example k products $u_i \cdot w_{ij}$ must be accumulated and the result x must be stored in a data word of width $b_x < (b_u + b_w + \mathrm{ld}\, k)$. Here an appropriate result window must be selected from the long accumulator. The problems of *rounding* (at the lower end of the window) and *saturation* (at the upper end of the window) must be solved and should be realized efficiently by hardware. In case of

example 2 the selection of a result window must be performed after multiplying the two fixed point numbers u and δ (see Fig. 2). So a special component in the neuroprocessor design (called *normalizer*) is required for the selection of result windows (including support of rounding and saturation).

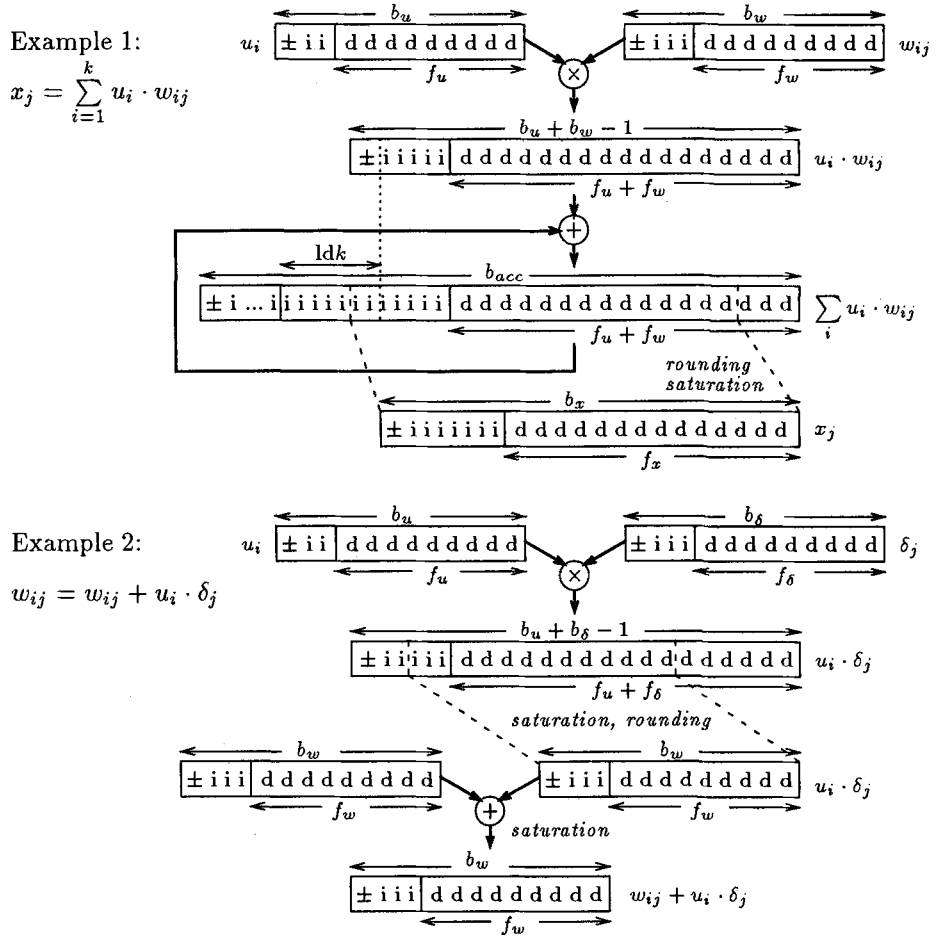

Fig. 2. Fixed point realizations of important neural operations

Besides of the normalizer, also a multiplier, an accumulator and a second adder (for calculating Euclidean distances) should be available in the neuroprocessor design. Fig 3 shows an arrangement of all components which is suitable for all neural operations listed above. To support the required configurability all shaded components must operate in parallel on a variable number of p fixed point data elements coded in one very long data word.

Two data words – each containing p data elements of b/p bits – enter the arithmetical unit at the input ports S and W. The adder may be used to calculate in parallel p differences. It also must support saturation because each result must be encoded again in b/p bits and so an overflow condition may occur. The multiplier performs p parallel multiplications of two data elements of b/p bits and produces p results of $2 \cdot b/p$ bits encoded in a long word of $b_{mult} = 2b$ bits.

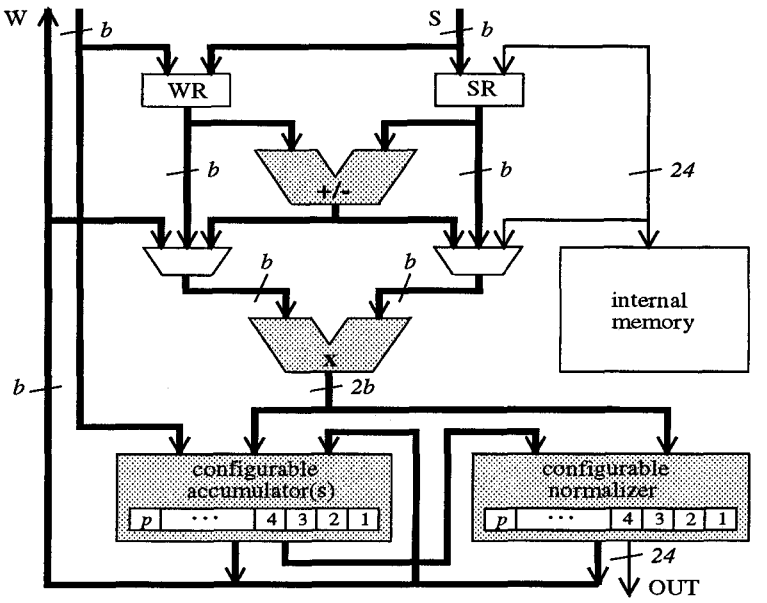

Fig. 3. The arithemetical unit

The accumulator must perform p parallel accumulations. The total length b_{acc} of the accumulator should exceed b_{mult} by $p_{max} \cdot \mathrm{ld}\, k$ bit positions to allow a large number of k additions without overflow. For certain neural network operations (see also Sect. 3) some parameters must be stored in an additional local memory. The parameters are identical for all p parallel computations, so a memory width of 24 bit is sufficient. Two outputs are required:

- a scalar output of 24 bits is used e.g. for delivering the result of acccumulations, and
- a parallel output of b bits for storing p results encoded in one very long data word (here one of the input ports can serve also as an output port).

In order to study the cost of configurability a prototype neuroprocessor chip has been designed by using VHDL. A width of $b = 48$ bits was selected for very long data words, the parallelism degree p can be varied from $p_{min} = 2$ to $p_{max} = 12$. So two 24 bit data elements, three 16 bit data elements, ..., or twelve 4 bit data elements can be encoded in one very long data word.

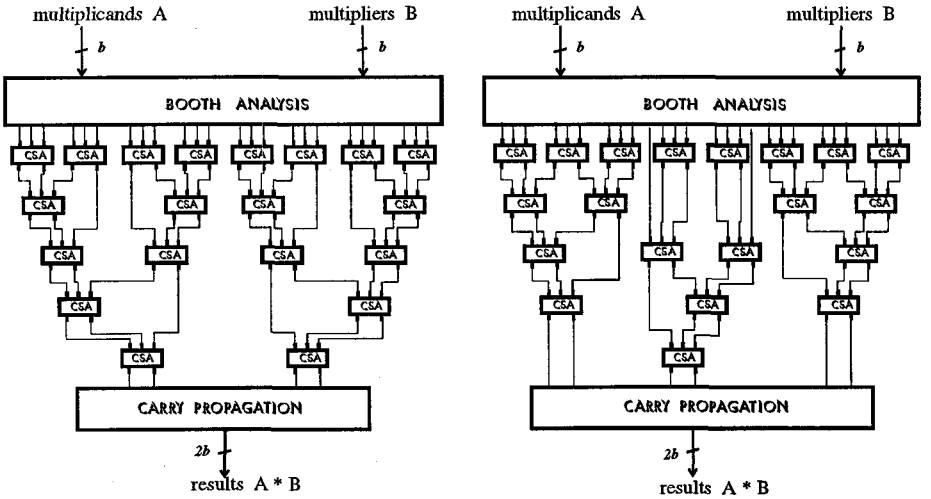

Fig. 4. Concept for the realization of a configurable multiplier

The configurable multiplier can perform two 24 × 24 bit, three 16 × 16, ..., twelve 4 × 4 bit parallel multiplications. It is realized by introducing multiplexers in the adder tree because the CSA adders must be arranged in a different way for different configurations (see Fig. 4). However the Booth analysis always remains unchanged. The configurable adder can be realized by a standard adder operating on long words of $b + p_{max}$ bits. The p elementary data words encoded in one very long data word are separated by zero bits to prevent a carry signal from being propagated into the neighbour field (see Fig. 5).

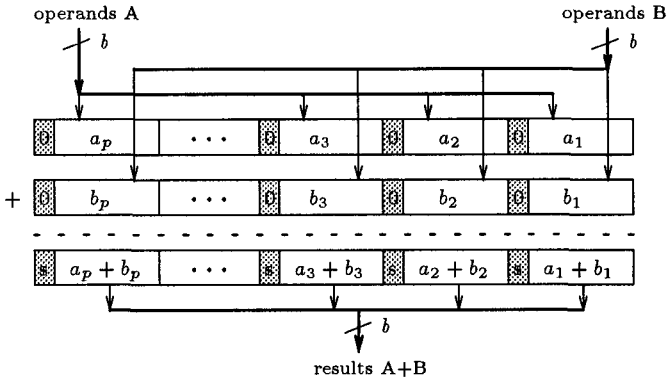

Fig. 5. Concept for the realization of the configurable adder

The complete neuroprocessor chip including also address generators, control logic and local memory of size 512 × 24 has been synthesized by using VHDL and standard cells in 0.7μ technology [5]. All cricital components like adder, multiplier and accumulator have been implemented by structural descriptions. The required area (including configurable and standard components) is approx. 85 mm^2. Table 1 summarizes in the first column the relative chip area used by the arithmetic components, the second column presents the relative component area that is only used for implementing the configurability.

Table 1. The cost of configurability for all arithmetic components (in chip area):

component	chip area	component area used for configurabilty
I/O ports	3.2 %	38 %
adder	16.9 %	18 %
multiplier	39.5 %	13 %
accumulator	20.3 %	18 %
normalizer	20.1 %	40 %

It can be seen that for the most complex arithmetical units, the multiplier and the accumulator, the additional cost of configurability is rather small.

3 Implementation of Neural Network Operations

A variable number of n neuroprocessors can operate in parallel to simulate a neural network. The input/output W of each neuroprocessor is connected to a local weight memory (WM), the input S is connected to a global state memory (SM) where the input/output patterns are stored. So in each clock cycle each neuroprocesoor can receive and operate on two very long data words.

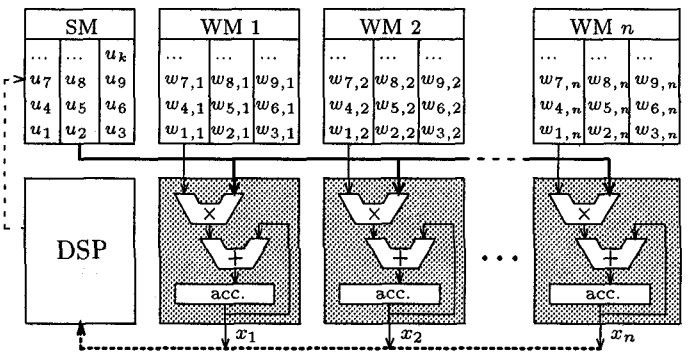

Fig. 6. Implementation of the neural operation $x_j = \sum_{i=1}^{k} u_i \cdot w_{ij}$ for $p = 3$

All n neuroprocessors are controlled by a DSP which also addresses the state memory and performs all not computation-intensive operations (like the computation of neural transfer functions). Fig. 6 shows how a vector-matrix-multiplication $(u_1, u_2, \ldots, u_k) \cdot W_{k \times n}$ is implemented on a system with n neuroprocessors. The parallelism degree is set to $p = 3$, so (u_i, u_{i+1}, u_{i+2}) and $(w_{i,j}, w_{i+1,j}, w_{i+2,j})$ are stored in very long memory words of SM and WM j and processed in parallel by processor j. The results x_j are read sequentially from the OUT ports of all processors by the DSP while the n neuroprocessors accumulate the next n elements of $\mathbf{x}$.

In case of gradient-based learning algorithms in multilayer neural networks often the operation $x_i = \sum_{j=1}^{m} w_{ij} \cdot \delta_j$ must be realized which represents a multiplication of an error vector $\boldsymbol{\delta}$ with a transposed matrix W^T. Depending on m, p and n this operation can be realized efficiently in two different ways:

1. *for small networks ($m < p \cdot n^2$):* store additionally $w^T_{i,j} = w_{j,i}$ in WM j and compute $x_j = \sum_i \delta_i \cdot w^T_{ij}$ (see previous operation). In this case neuroprocessor j must compute each weight update twice (for $w_{i,j}$ and $w^T_{i,j}$).
2. *for large networks ($m \geq p \cdot n^2$):* store $w_{i,j}$ only in WM j and distribute the vector $\boldsymbol{\delta}$ among all *internal* memorys. Each of the n neuroprocessors computes the products $\delta_j \cdot (w_{i,j}, w_{i+1,j}, \ldots, w_{i+p-1,j})$ for all locally stored values δ_j and accumulates the p products in p accumulators. Thereafter, the $n \cdot p$ partial sums $x_i, x_{i+1}, \ldots, x_{i+p-1}$ are read and added by the DSP (see Fig. 7) while the n processors start accumulating the next p elements of $\mathbf{x}$.

Finally, Fig. 8 illustrates the parallel computation of a typical weight update operation bases on the outer vector product.

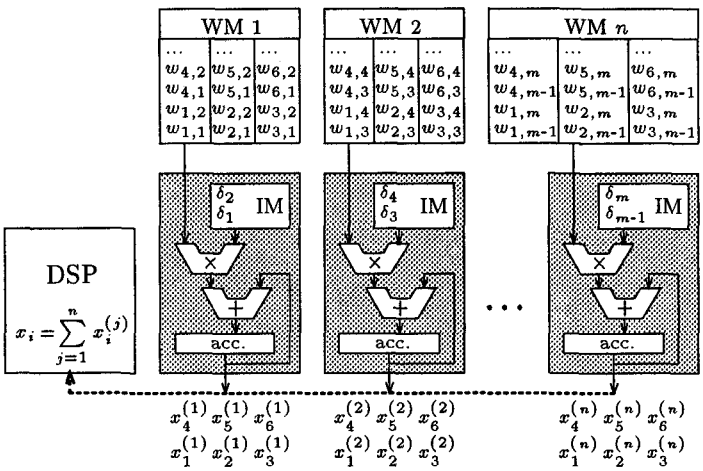

Fig. 7. Implementation of the neural operation $x_i = \sum_{j=1}^{m} w_{ij} \cdot \delta_j$ for $p = 3$

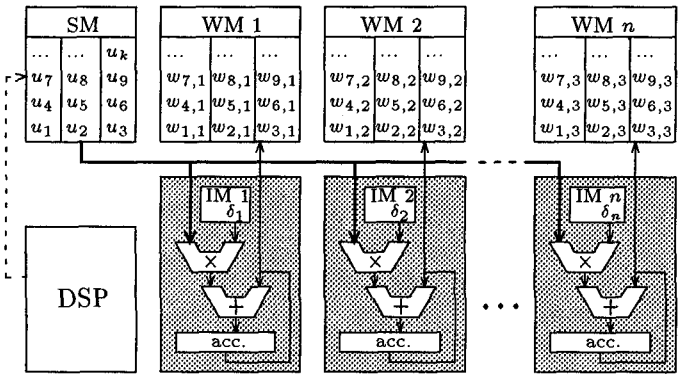

Fig. 8. Implementation of the neural operation $w_{ij} = w_{ij} + u_i \cdot \delta_j$ for $p = 3$

4 Conclusion

In this paper a new concept for neurocomputer architectures has been presented. It is based on a configurable neuroprocessor design that can be adapted dynamically to different data precisions required for the simulation of different neural network models. The design of a prototype has demonstrated that configurabilty can be realized with low additional cost.

References

1. D. Hammerstrom. A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning. In *Proc. IJCNN*, pages 537–543, San Diego, 1990.
2. J. Holt and J. Hwang. Finite precision error analysis of neural networks hardware implementations. *IEEE Trans. on Computers*, 42:281–290, 1993.
3. P. Ienne. Architectures for Neurocomputers: Review and Performance Evaluation. Technical Report 93/21, Swiss Institute of Technology, Lausanne, 1994.
4. U. Ramacher. Synapse – A Neurocomputer that Synthesizes Neural Algorithms on a Parallel Systolic Engine. *Journal of Parallel and Distributed Computing*, 14:306–318, 1992.
5. A. Strey, N. Avellana, R. Holgado, J.A. Fernández, R. Capillas, and E. Valderrama. A Massively Parallel Neurocomputer with a Reconfigurable Arithmetical Unit. In J. Mira and F. Sandoval, editors, *From Natural to Artificial Neural Computation*, pages 800–806. Springer, Berlin Heidelberg, 1995.
6. P. Thiran, V. Peiris, P. Heim, and B. Hochet. Quantization effects in digitally behaving circuit implementations of Kohonen networks. *IEEE Trans. on Neural Networks*, 5(3):450–458, 1994.
7. M.A. Viredaz. MANTRA I: An SIMD Processor Array for Neural Computation. In P.P. Spies, editor, *Euro-ARCH '93*, pages 99–110. Springer-Verlag, 1993.