A Vector Orthogonal Multiprocessor NEOMP and its Use in Neural Network Mapping

Jose Hiroki Saito
Grupo de Arquitetura, Processamento de Imagens e Sinais
Universidade Federal de São Carlos
Rodovia Washington Luis, Km 235, 13565-905 São Carlos – SP , Brasil
{e-mail: saito@dc.ufscar.br }

Abstract –

A vector Orthogonal Multiprocessor architecture NEOMP and its use by a feedforward artificial neural network, Neocognitron, is described. The proposed architecture is composed by several vector processing units, and a scalar control processor, which access the memory modules in a orthogonal fashion. The performance analysis of the architecture is realized, identifying the concurrent computation grains in the neocognitron, which are attributed to the vector processors. The analysis of the architecture showed that its speed-up is linear in a wide range, where the implementation of NEOMP is appropriate. The scalar control processor and the vector processing unit hardware prototype were simulated and showed the feasibility of their implementation, each one in a single FPGA, which inspire the construction of NEOMP as a real time neocognitron, and other feedforward neural network systems, using vector orthogonal multiprocessor architecture.

Keywords – **orthogonal multiprocessor, OMP, vector processor, parallel processing, neocognitron, neural network, NEOMP**

I. INTRODUCTION

The Orthogonal Multiprocessor, *OMP*, is characterized by the parallel processing units, each one accessing their memory modules in two ways: column, and row. Each row, and each column, is attributed to one exclusive processor. The row access and column access are performed exclusively, without time sharing of the buses by the multiple processors, without memory access conflits.

Meanwhile, an artificial neural network involves concurrent operations of a great number of artificial neurons. These neurons have a great number of weighted input connections. Several hardware implementation of artificial neural networks have been proposed, but most current implementations are software implementations, which are time demanding, particularly for the learning phase [AMA 97][KUM 94][YAS 98].

The proposed architecture NEOMP uses several vector units, and a scalar control processor, accessing the orthogonal memory modules, to provide the input data to the weighted sum operations of the neural network, and to optimize the processing time.

NEOMP presents an interesting solution to the neuronal data exchange after one layer processing, in a feedforward network implemented in parallel architecture. A shortcoming of an OMP is the excessive increase of the number of memory modules, when it is increased the number of processors, so that the architecture becomes well fitted to small number of processors. Another shortcoming is the prohibition of mixed row and column access of the orthogonal memory, which reduces the flexibility of the mapping of the algorithms, which do not access the memory in orthogonal fashion.

The performance analysis of NEOMP is focusing the parallel execution aspects of an artificial neural network, neocognitron, proposed initially to handwritten character recognition, by Fukushima [FUK 79]. Fortunately, neocognitron is characterized by the predominance of concurrent execution of compound vector functions, to all cell-planes, which makes effective the use of orthogonal memory. Similar hardware implementation extends to other neural network models .

The scalar control unit prototype, and the vector processing unit prototype of NEOMP were simulated and showed the feasibility to integrate the majority of the vector operations in a single FPGA, which enables the implementation of a simple hardware real time neocognitron system, with a few number of the vector units. The following sections are related to NEOMP description and its performance analysis by the use of the neocognitron algorithm. Then follows some aspects of the hardware prototype and conclusion.

II. NEOMP ARCHITECTURE

An Orthogonal Multiprocessor architecture [HWA 89][HWA 93] is composed by p processors, which access simultaneously p rows or p columns of interleaved p^2 memory modules, which are interconnected by p row buses and p column buses. The row access and column access are performed exclusively, without time sharing of the buses by the multiple processors, reducing the memory access

conflits. Each processor access exclusively one row bus and one column bus, so that one memory module is accessed by two processors at most. If we denote a memory module by M_{ij}, this module may be accessed by processor i, in a row mode, and by processor j, in a column mode. A memory module M_{ii}, which is the diagonal memory module, is accessed exclusively by processor i.

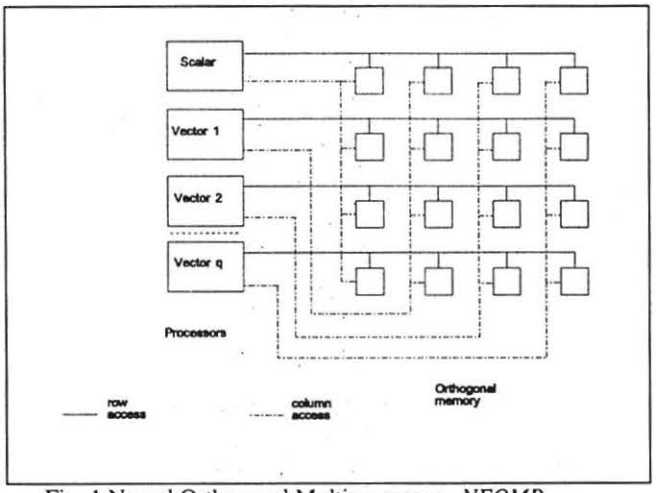

Fig. 1 Neural Orthogonal Multiprocessor - *NEOMP*

An *OMP* architecture is efficient to several scientific computations, but the orthogonal memory access principle prohibits memory access in mixed modes. This may reduce the flexibility of the mapping of the algorithms, which do not access the memory array in orthogonal fashion.

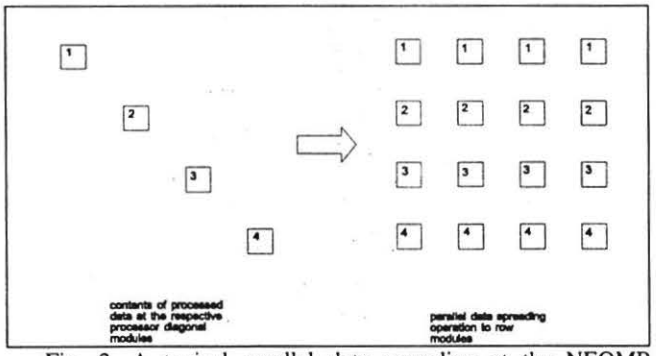

Fig. 2. A typical parallel data spreading at the NEOMP memory modules.

We proposed an orthogonal multiprocessor to Neural Network Mapping Application, *NEOMP*, with q vector processing units working synchronized, and a scalar processor responsible to control the vector units, all of them connected to the orthogonal memory . Fig. 1 shows the proposed orthogonal multiprocessor diagram. The scalar processor is also responsible by the sequencial execution portion of algorithms. Note that the total number of processors of NEOMP architecture is $p = q + 1$, considering the scalar processor.

Fig. 2 shows a typical data spreading operation using the NEOMP memory modules. At the left side we can see the diagonal memory modules which correspond to local memory of the processors, with their respective data . With a row access, the processors connected to their exclusive rows spreads the data to all row modules, simultaneously. At the right side, we see the result of parallel data spreading operation, with all columns containing the same data, distributed in several memory modules. During the following column access, all processors may access their own column memory modules, which contain the spreaded data.

III. NEOCOGNITRON

Neocognitron is a massively parallel neural network, composed of several layers of neuron cells, proposed by Fukushima [FUK 79] [FUK 82] [FUK 92] [FUK 96] [SAI 98] inspired by Hubel and Wiesel's model of biological vision. As other neural network models, neocognitron has its self organized training phase and the recognition phase, when used in pattern recognition.

```
Program neocognitron ();
begin
   For l = 1 to L do     compute_stage (l);
end;
```

Fig. 3. Neocognitron computation sequence.

The lowest stage of the network is the input layer U_0. Each of the succeeding i-th stages has a layer U_{Si} consisting of S-cells followed by a layer U_{Ci} of C-cells. Each layer of S-cells or C-cells is composed by a number of two-dimensional array of cells, called *cell-planes*. Each cell-plane is associated to a single feature extracted from the training patterns, during the learning phase. The S-cells are responsible by the feature extration, and the input connection weights of the S-cells are adjusted during the training phase. C-cells are responsible by the distorted features correction. Each cell inside a cell-plane receives input connections from the preceding layer cell-planes.

The recognition phase of neocognitron, follows the sequence showed at the following algorithm, Fig.3, from layer 1 to L, where L is the number of stages.

Fig.4 shows the algorithm to compute the output value $u_{Sl}(n, k)$ of a S-cell, and the output value $u_{Cl}(n, k)$ of a C-cell, from the stage l. It is computed the output values to all K_l cell-planes and all N cell-positions inside the cell-plane.

To obtain the $u_{Sl}(n, k)$ value, it is computed the weighted sum, $e(n,k)$ and $h(n,k)$, of all inputs coming from all K_{l-1} cell-planes of the preceding layer, in a given

connection area S_v, which surrounds the position n, of the preceding layer C-cell, or input layer, by the following equations (1) and (2)

$$e(n,k)= e(n,k)+a(\nu,\kappa,k).u_{Cl-1}(n+\nu,\kappa), \text{ and} \quad (1)$$

$$h(n,k)= h(n,k)+c(\nu).\{u_{Cl-1}(\kappa, n+\nu)\}^2 . \quad (2)$$

Then the $u_{Sl}(n, k)$ value is obtained by the equation :

$$u_{Sl}(n,k)=(\theta/(1-\theta)) . \varphi((1+ e(n,k))/(1+\theta.b(k).sqrt(h(n,k)))-1), \quad (3)$$

where

$\varphi(x) = x$, when $x > 0$, and $\varphi(x) = 0$, elsewhere. The variable θ represents the threshold of the function, whose value is between 0, and 1, and $b(k)$ represents the inhibition coeficient.

```
Procedure compute_stage (l) ;

Begin
 For k = 1 to K_l do begin
   for n = 1 to N do begin
     for κ = 1 to K_{l-1} do
       for all ν ε S_ν do begin
          e(n,k)= e(n,k)+a(ν,κ,k).u_{Cl-1}(n+ν,κ) ;
          h(n,k)= h(n,k)+c(ν).{u_{Cl-1}(κ, n+ν)}^2 ;
       end;
       u_Sl(n,k)=(θ/(1-θ)) .
       φ((1+ e(n,k))/(1+θ.b(k).sqrt(h(n,k)))-1) ;
       for all ν ε S_ν do
        u_Cl(n,k) = u_Cl(n,k) + d(ν).u_Sl(n+ν, k);
       u_Cl(n,k) = Ψ (u_Cl(n, k))
   end
 end;
end;
```

Fig. 4. Algorithm to compute the S-cells and C-cells of the l-th stage.

To obtain $u_{Cl}(n, k)$, it is computed the weighted sum of all inputs corresponding to the previously obtained $u_{Sl}(n, k)$, in a given connection area S_v, which surrounds the position n, of the preceding S-cell layer, by the following equation:

$$u_{Cl}(n,k) = u_{Cl}(n,k) + d(\nu).u_{Sl}(n+\nu, k), \quad (4)$$

followed by calculation of transfer function $\Psi(x) = \varphi(x) / (1 + \varphi(x))$, which limits C-cell output to the range $[0,1)$.

At the last stage, the computation of the C-cells are a little modified, because they must be conditioned to the categories of patterns. We will not detail that in this paper because it does not affect the proposal of the architecture.

IV. NEOCOGNITRON TRAINING

The training phase follows the algorithm of Fig. 5. At the stage l, the training proceeds, as showed at the algorithm of Fig.6, computing the S-cell values to cell-planes, $k = 1$ to $k = K_l$, related to already existing features, and $k = K_l + 1$, which corresponds to the seed selecting plane. Then it is verified the seed selecting plane, by the cell responding most strongly, called *winner*. If at the *winner* position there is any response greater than zero at the previous K_l cell-planes, the algorithm proceeds searching for the next *winner*; otherwise, each input connection of the seed cell is reinforced proportionally to the intensity of the input connection cell, as the equations :

$$a(\nu,\kappa,k)= a(\nu,\kappa,k) + q.c(\nu).u_{Cl-1}(winner+\nu,\kappa), \text{and} \quad (5)$$

$$b(k) = b(k) + q. sqrt(h(winner,k)). \quad (6)$$

```
Program train_neocognitron();

begin
  for l = 1 to L do train_stage (l);
end;
```

Fig. 5 Neocognitron training sequence.

```
procedure train_stage (l);
begin
 repeat
    for k = 1 to K_l + 1 do compute_stage(l);
    selected =false;
    repeat
      if next_winner> 0 then begin
          winner = next_winner;
          selected = true;
       for k = 1 to K_l do
            if u_S (winner, k) > 0 then
                    selected =false;
      end;
    until (selected or next_winner = 0);
    if selected then begin
     for κ = 1 to K_{l-1} do
       for all ν ε S do
          a(ν,κ,k)= a(ν,κ,k) +
                  q.c(ν).u_{Cl-1}(winner+ν,κ) ;
      b(k) = b(k) + q. sqrt(h(winner,k)) ;
      K_l = K_l +1
    end
  until not (selected);
end;
```

Fig. 6. Neocognitron training algorithm.

By this way a new feature is extracted and a new cell-plane is added to the layer, incrementing K_l by one.

The training process within a S -layer, described above, is repeated until all new features are detected at the seed selecting plane, with the presentation of training patterns at the input layer.

V. CONCURRENT COMPUTING GRAINS

By the analysis of the neocognitron model described at previous section we extracted a set of concurrent computation grains, which corresponds to compound vector functions [HWA 93]. Four grains *E*, *H*, *S*, and *C*, correspond to the neocognitron computing algorithm; and three grains *F*, *Ra*, and *Rb*, are related to the training algorithm.

***E*-grain** computes to all N positions of the cell-plane, the weighted sum of the input $u_{Cl-1}(n+\nu,\kappa)$, with the weight $a(\nu,\kappa,k)$, within the connection region S_ν, which results in a parcial value of $e(n,k)$, corresponding to the plane κ. Each *E*-grain issue repeats the flow graph N times, to compute all positions of the E matrix. Its result is added to the E matrix which will accumulate the weighted sum of all κ cell-planes, after κ issues of the *E*-chain function.

***H*-grain** is similar to the *E*-grain but the weight values are $c(\nu)$ and the input values are squared before weighted sum. It results in the H matrix, which will contain the weighted sum of all preceding layer cell-planes, after κ issues of the *H*-grain function.

***S*-grain** computes $u_{Sl}(n,k)$, to all N positions of the cell-plane, using the results of the previously described *E*-grain and *H*-grain functions.

***C*-grain** corresponds to the computation of the $u_{Cl}(n,k)$, to all N cells. It computes the weighted sum of $u_{Sl}(n+\nu, k)$, by $d(\nu)$, and then the function Ψ.

***F*-grain** corresponds to the feature extraction during the training phase. It is used to detect a new feature address, by the use of the seed selecting plane concept. When it is computed the *winner*, it is verified if there is any non-zero value at the same address, at the previous cell-planes of the same layer. If any non-zero value is present at the same address of previous cell-planes, the winner address is used by *Ra*-grain and *Rb*-grain functions.

***Ra*-grain** is used to reinforce the weight values $a(\nu,\kappa,k)$ during the training algorithm. Each *Ra*-grain function issue reinforces the weight values $a(\nu,\kappa,k)$, which correspond to the κ-th cell-plane of the preceding layer.

***Rb*-grain** is used to compute the reinforcement of the $b(k)$, values. Note that this function is independent of the preceding layer.

We classified the grain size as the table I, which shows at the first column the identified grains; at the second column, the number of arithmetic operations per issue of the processing grain; followed by the complexity of the processing, in O function; at next column it is showed the typical number of arithmetic operations, using $N = 400$, $S_\nu = 25$, and $K = 50$; and finally, the processing time, using a processor cycle of $\tau = 100$ ns.

It is considered that the arithmetic operations are executed sequencially within the grain, disregarding the instruction and operands memory access overheads. We are not considering here the pipelined execution, or vector processing, which may improve the processing time of the majority of the grains.

TABLE I
COMPUTING GRAINS

Grain	Operations per issue	O( )	Typical Values	Execution Time ($\tau = 100$ ns)
E	$2.N.S_\nu + N$	$N.S_\nu$	20.400	2.04 ms
H	$N.S_\nu.3 + N$	$N.S_\nu$	30.400	3.04 ms
S	$N.4$	N	1.600	0.16 ms
C	$N.S_\nu.2$	$N.S_\nu$	20.000	2.00 ms
F	$N + K$	$N.K$	20.000	2.00 ms
Ra	$S_\nu.3$	S_ν	75	7.5μs
Rb	3	-	3	0.3μs

VI. PERFORMANCE ANALYSIS

As an example, the mapping of the neocognitron to the proposed architecture may be resumed, as follows. The grains *E*, *H*, *S*, and *C*, are processed at the vector processors, and the other functions at the scalar processor.

We can analyse one stage processing of neocognitron. A stage of K_l cell-planes, and K_{l-1} preceding cell-planes, will compute $K_l.K_{l-1}$ times the grains *E*, and *H;* and K_l times the grains *S*, and *C*. If the number of vector units is q, then we can execute the grains *E*, and *H*, in $\lceil K_l.K_{l-1}/q \rceil$ steps, and then the *S*, and *C*, grains in $\lceil K_l/q \rceil$ steps. We observe that the expression inside vertical bars used in this paper means minimum integer greater or equal their value. The total processing time of one stage is then:

$$T_q = \lceil K_l.K_{l-1}/q \rceil . T_E + \lceil K_l.K_{l-1}/q \rceil . T_H + \lceil K_l/q \rceil . T_S + \lceil K_l/q \rceil . T_C \quad (7)$$

If we consider that T_E, T_H, and T_C, are of the same complexity, we can rewrite the equation (7), as:

$$T_q = (2. \lceil K_l.K_{l-1}/q \rceil + \lceil K_l/q \rceil). T_E + \lceil K_l/q \rceil . T_S \quad (8)$$

If we regard the memory data exchange time, we need to consider that K_{l-1} cell-planes are broadcasted to q vector units memory modules, in a orthogonal fashion. It means that q cell-planes are distributed simultaneously, so that the total memory data exchange overhead value is $\lceil K_{l-1}/q \rceil$ times one cell-plane distribution time T_M, which is $N.q.\tau_c$, where τ_c is the memory access time. Before the data spreading operation, each processor owns its diagonal module data. After that, each processor can access all spreaded data by column access.

Now the equation (8) may be rewritten as follows:

$$T_q = (2. \lceil K_l.K_{l-1}/q \rceil + \lceil K_l/q \rceil). T_E + \lceil K_l/q \rceil . T_S + \lceil K_{l-1}/q \rceil . T_M \quad (9)$$

The processing time to one processor T_1 will be computed as follows,

$$T_1 = (2.K_l . K_{l-1} + K_l). T_E + K_l . T_S. \quad (10)$$

which may result in

$$T_1 = K_l . [(2.K_{l-1} + 1) . T_E + T_S.]. \quad (11)$$

If $T_E = O(N.S_v)$ and $T_S = O(N)$, as table 1, we substitute at (11) the T_S, to obtain:

$$T_1 = K_l . (2.K_{l-1} + 1 + 1/S_v) . T_E , \quad (12)$$

and if $K_{l-1} >> 1$, we should rewrite the equation as:

$$T_1 = K_l . 2 . K_{l-1} . T_E . \quad (13)$$

The speed-up S_q should be calculated by:

$$S_q = T_1 / T_q . \quad (14)$$

The speed-up is affected by the memory data distribution overhead, and the idle vector units, during the T_q processing time computing. If we consider that $q << K_{l-1}$ and $q << K_l$, we reduce the number of idle vector units. In this case, $|K_{l-1} / q|$, and $|K_l / q|$, may be approximated to K_{l-1} / q, and K_l / q, respectively, and S_q results in:

$$S_q = q / (1 + T_M / (2.K_l . T_E)) , \quad (15)$$

which is rewritten as:

$$S_q = q / (1 + \alpha) , \quad (16)$$

where $\alpha = T_M / (2.K_l . T_E)$.

By equation (16) we see that α should be smaller than possible, to achieve the linear speed-up.

We can analyse α, calculating the O function, as follows:

$$O(\alpha) = O(T_M / (2.K_l . T_E)), \text{ so}$$

$$O(\alpha) = O(N.q / (2.K_l . N.S_v)), \text{ and}$$

$$O(\alpha) = O(q / (2. K_l . S_v)) . \quad (17)$$

We considered above that $q << K_l$, so by the equation (17) we can conclude that α is negligible unless we increase excessively the number of vector processing units, which is not our initial proposal.

VII. MAPPING RESULT

It follows the computation result of the mapping of the architecture to a tipical neocognitron used in pattern recognition. Table II shows the network structure, of a typical neocognitron structure, with five stages (layers) composed by S-cells and C-cells columns.The rows are showing the cell-plane dimension N, connection area S_v, number of cell-planes, and the number of grain execution issues I_E, I_H, I_S, and I_C, during the recognition phase, and I_F, I_{Ra}, and I_{Rb}, during the training phase. Table III shows the sequencial processing time, at respective stages, and the table IV, the total processing time, and the speed-up, using the processor cycle time of $\tau = 100$ ns. The rows are varying the number of vector processing units, from 1 to 32. Here we consider the memory data spreading time with memory access time of $\tau_c = 100$ ns. The speed-up obtained are very close to the number of vector units. If it is regarded the scalar control unit, the speed-up is close to linear, which is the ideal case. Table V shows the neocognitron training time by stage, regarding that during the training process, the feature extraction procedure is repeated several times, varying the input patterns. The values showed are minimum because those values corresponds to the processing time when the features was actually extracted. The training time is reduced increasing the number of vector processors because the grains E, H, S, and C, executed during the recognition phase, are also used repeatedly at each feature extraction operation.

TABLE II
NEOCOGNITRON STRUCTURE AND COMPUTING GRAIN EXECUTION

	Stage 1		Stage 2		Stage 3	
grain	S1	C1	S2	C2	S3	C3
N	57x57	57x57	57x57	21x21	21x21	13x13
S_v	5x5	3x3	5x5	5x5	7x7	5x5
K	16	16	16	16	62	62
I_E	16	-	256	-	992	-
I_H	16	-	256	-	992	-
I_S	16	-	16	-	62	-
I_C	-	16	-	16	-	62
I_F	16	-	16	-	62	-
I_{Ra}	16	-	16	-	62	-
I_{Rb}	16	-	16	-	62	-

TABLE II (cont.)

	Stage 4		Stage 5	
grain	S4	C4	S5	C5
N	13x13	7x7	7x7	3x3
S_v	7x7	5x5	5x5	3x3
K	99	99	96	96
I_E	6,138		9,504	
I_H	6,138		9,504	
I_S	99		96	
I_C		99		96
I_F	99		96	
I_{Ra}	99		96	
I_{Rb}	99		96	

Table VI shows the neocognitron total training time, considering the training time by stage showed at the previously showed Table V.

Table VII shows the memory data spreading time, when the number of processors are varying from 1 to 32. The memory access time used is $\tau_c = 100$ ns. It is noted that the spreading time increases with the number of vector processors. When the number of processors is double, the data spreading time is also doubled.

Fig.7 shows the speed-up diagram using the data of table IV. There we can note that the speed-up is close to linear along the range of 1 to 16 processors, and after that, degrades slowly, until 32 processors.

A good implementation of NEOMP may vary from 2 to 8 processors, where the number of memory modules varies, not excessively, from 4 to 64, considering the typical values of the neocognitron structure, as showed at table II.

TABLE III
STAGE PROCESSING TIME

q	Stage 1 (ms)	Stage 2 (ms)	Stage 3 (ms)	Stage 4 (ms)	Stage 5 (ms)
1	774	10,619	10,840	25,652	5,917
2	393	5,314	5,420	12,826	2,958
3	299	3,574	3,626	8,550	1,972
4	205	2,659	2,717	6,415	1,479
5	209	2,165	2,180	5,132	1,183
6	164	1,789	1,819	4,275	986
7	167	1,543	1,555	3,665	845
8	121	1,332	1,358	3,209	739
16	99	668	679	1,606	369
32	151	343	340	803	184

TABLE IV
TOTAL PROCESSING TIME AND SPEED-UP

q	Total (s)	Speed-up	q	Total (s)	Speed-up
1	53.8	1	6	9.0	5.95
2	26.9	1.99	7	7.7	6.91
3	18.0	2.98	8	6.7	7.96
4	13.4	3.99	16	3.4	15.72
5	10.8	4.95	32	1.8	29.54

VIII. HARDWARE SIMULATION

When it is considered the hardware implementation of neural networks, with the current technologies, the first question which arises is the use of analog and digital technology [IWA 95]. The analog circuits have the following characteristics: (a) current and voltages allows the implementation of arithmetic operators; (b) the circuit dimension is reduced; (c) weak to noisy immunity, and to the construction of complex and large circuits. At the other hand the digital circuits are characterized as: (a) strong to noisy immunity, and to the construction of complex and large circuits; (b) easy to memory implementation, and processing of large amount of data. We conclude that the digital technology is adequated to the implementation of large neural networks, using neurochips, improved microprocessors, or dedicated digital signal processors. Field Programmable Gate Arrays (FPGA's) have been become an important technology for the design of VLSI circuits and systems. The field programmability of the components leads to fast implementation of application-specific integrated circuits, rapid prototyping, circuit emulation, reconfigurable circuit and system design [HER 98][LEW 98][TSU 98].

The use of FPGA's is also suited to the EHW (Evolvable HardWare) context of project. It is notorious that the traditional hardware is inflexible. EHW is a new field whose architecture, structure, and functions change dynamically and autonomously in order to improve its performance [YAO 99].

TABLE V
STAGE TRAINING TIME (s)

q	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
1	6	90,261	341,460	1,282	286
2	3	45	170	641	143
3	2	30	114	427	95
4	1	22	85	320	71
5	1	18	68	256	57
6	1	15	57	213	47
7	1	13	48	183	40
8	1	11	42	160	35
16	1	5	21	80	17
32	1	2	10	40	8

TABLE VI
TOTAL TRAINING TIME (min,s)

q	Total	q	Total	q	Total
1	33' 27''	5	6'42''	16	2'6''
2	16' 43''	6	5'34''	32	1'3''
3	11' 9''	7	4'46''		
4	8' 21''	8	4'10''		

We projected the vector unit prototype using an FPGA environment. As a sample prototype the grains *E, H, S,* and *C,* were implemented in 8 bits integer input data, and a timing unit was included to enable the complete execution of the functions. The square root and divide operations were not included, at this first prototype. All other circuits were possible to fit in a single FPGA, EPF10K40, with 40.000 gates, which showed the feasibility to construct hardware prototypes of real time neocognitron systems, with a few number of components. Although the complete prototype involves the orthogonal memory system and the scalar processor unit, the number of memory modules will not be so great if the number of vector units is small in the proposal.

The scalar processing unit prototype was also projected using the same FPGA environment. It is a simple fixed control processor with small number of instructions, as a RISC processor of 16 bits data. The instruction level parallelism is also explored, making possible the interpretation of two independent instructions

simultaneosly, at the scalar processor, which functions as a superscalar processor. The corresponding circuit was feasible to construct in a single FPGA, as the vector processing unit.

TABLE VII
MEMORY DATA SPREADING TIME (μs)

q	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
1	-	-	-	-	-
2	6,498	5,198	705	270	78
3	9,747	5,848	793	304	88
4	12,996	5,198	705	270	78
5	16,245	6,498	882	338	98
6	19,494	5,848	793	304	88
7	22,743	6,822	926	354	102
8	25,992	5,198	705	270	78
16	51,984	5,198	705	1,081	548
32	103,968	10,396	1,411	1,081	627

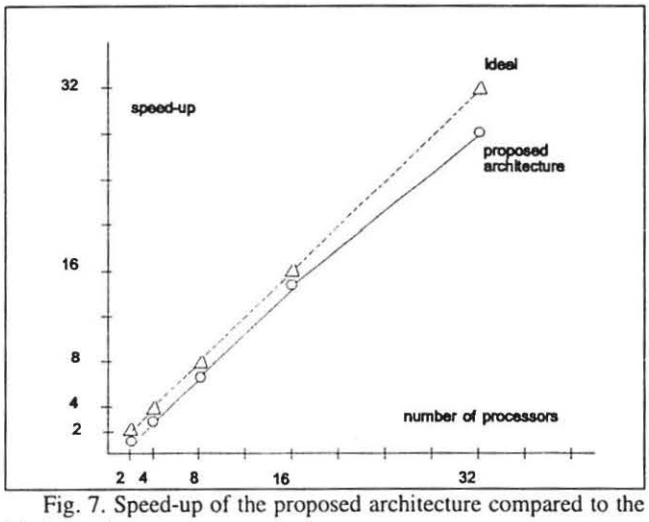

Fig. 7. Speed-up of the proposed architecture compared to the ideal speed-up.

IX. CONCLUSION

Orthogonal mutiprocessor architecture is adequated to a special scientific computation where the data processing may be parallelized, besides of their frequent and volumous interchange. In this way, a typical application is the hardware implementation of neural network models. Neocognitron is a known model of neural network which explicits their internal interconnections layer by layer, and it consists a good mapping example, in the performance analysis of the proposed vector orthogonal multiprocessor NEOMP.

By the above result we can conclude that the proposed arquitecture is suitable to the neocognitron mapping, when the number of vector functional units is not considerable great, compared to the number of cell-planes of the neural network $q << K_l$, to all layers. By table IV, we note that the speed-up is near q even with $q = 32$, where $Sq = 29.54$. The total processing time to this number of functional units, is *1.8 s*, which qualifies the proposed architecture to real time application of neocognitron. The simulation of the hardware prototype showed that it is possible to construct eight bits vector units in a single FPGA, of 40.000 gates, as the control processor of sixteen bits. Although the complete prototype involves orthogonal memory system, it seems feasible to construct the hardware real time neocognitron system with a few number of FPGA components.

At the performance analysis presented at Section VI, it was considered the vector units sequencial processing time. If it is considered the pipeline operation, which is characteristic to vector processing units, the speed-up will be optimized. Another consideration is that the speed-up analysis was taken in comparation of one vector processor. Normally, at the software implementation of neocognitron, the processing time is reduced in comparation with the proposed vector processor unit.

Although the complete project and implementation of the NEOMP prototype is essential to the real evaluation of the architecture, it was showed in this work that the proposed architecture is suitable to the hardware implementation of feedforward neural network models, like neocognitron, using current technology.

ACKNOWLEDGMENT

The author would like to gratefully acknowledge Dr. K.Fukushima for the period of stay at the Osaka University.

REFERENCES

[AMA 97] AMAWY, A.El & LULASINGHE, P. Algorithmic Mapping of Feedforward Neural Networks onto Multiple Bus Systems, **IEEE Trans. On Parallel and Distributed Systems** Vol. 8, N.2, pp. 130-136, Feb. 1997.

[FUK 79] FUKUSHIMA, K., Neural-network model for a mechanism of pattern recognition unaffected by shift in position - neocognitron,**Trans. IEICE Japan**, vol. 62-A, no.10, pp. 658-665, 1979.

[FUK 82] FUKUSHIMA, K.& MIYAKE, S., Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shift in Position,

Pattern Recognition, vol. 15, no.6, pp.455-469, 1982.

[FUK 92] FUKUSHIMA,.K. & WAKE, N., Improved Neocognitron with Bend-Detecting Cells, **IEEE – International Joint Conference on Neural Networks**, Baltimore, Maryland, June 7-11, 1992, pp. 190-195, 1992.

[FUK 96] FUKUSHIMA,K.& TANIGAWA, M., Use of Different Thresholds in Learning and Recognition, **Neurocomputing**, 11, pp. 1-17, 1996.

[HER 98] HERZEN,B.V., Signal Processing at 250 MHz Using High-Performance FPGA's, **IEEE Trans. On VLSI Systems**, Vol. 6, N.2, pp. 238-246, Jun. 1998.

[HWA89] HWANG, K.; TSENG, P & KIM, D. , An Orthogonal Multiprocessor for Parallel Scientific Computations, **IEEE Trans. On Computers**, Vol.38,N.1,pp.47-61,Jan.1989

[HWA 93] HWANG,K.–**Advanced Computer Architecture–Parallelism,Scalability, Programmability**.McGrawHill,Sing., 1993.

[IWA 95] IWATA, A.& AMEMIYA, Y. – Neural Network LSI. **The Institute of Electronics, Information and Communication Engineers**, Japan, 1995.

[KUM 94] KUMAR, V.; SHEKHAR, S.& AMIN, M.B., A Scalable Parallel Formulation of the Backpropagation Algorithm for Hypercubes and Related Architectures, **IEEE Trans. On Parallel and Distributed Systems**, Vol. 5, N.10, pp. 1073-1090, Oct. 1994.

[LEW 98] LEWIS, D.M., GALLOWAY, D.R., IERSSEL, M., ROSE, J. & CHOW, P., The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System, **IEEE Trans. On VLSI Systems**, Vol. 6,N.2,pp. 188-198, Jun. 1998.

[SAI 98] SAITO, J.H. & FUKUSHIMA, K. , Modular Structure of Neocognitron to Pattern Recognition, **Proc. ICONIP'98, Fifth Int. Conf. On Neural Information Processing**, Kitakyushu, Japan, pp.279-282, Oct. 1998.

[TSU 98] TSUTSUI, A. & MIYASAKI, T., ANT-on-Yards: FPGA/MPU Hybrid Architecture for Telecommunication Data Processing, **IEEE Trans. On VLSI Systems**, Vol. 6, N.2, pp. 199-211, Jun. 1998.

[YAO 99] YAO, X., Following the Path of Evolvable Hardware, **Communications of the ACM**, Vol. 42, N.4, Ap. 1999.

[YAS 98] YASUNAGA, M., HACHIYA, I.; MOKI, K. & KIM, J., Fault Tolerant Self-Organizing Map Implemented by Wafer Scale Integration, **IEEE Trans. On VLSI Systems**, Vol. 6, N.2, Jun. 1998.