Presented at: DSA-92
Fourth Swedish Workshop on Computer System Architecture
Linköping, January 13th-15th, 1992

Designing Parallel Computers for Self Organizing Maps

Tomas Nordström

Division of Computer Science & Engineering
Department of Systems Engineering and Mathematics
Luleå University of Technology, Sweden
E-mail: tono@sm.luth.se

ABSTRACT

Self organizing maps (SOM) are a class of artificial neural network (ANN) models developed by Kohonen. There are a number of variants, where the self organizing feature map (SOFM) is one of the most used ANN models with unsupervised learning. Learning vector quantifiers (LVQ) is another group of SOM which can be used as very efficient classifiers. SOM have been used in a variety of fields, e.g. robotics, telecommunication and speech recognition.

Currently there is a great interest in using parallel computers for ANN models. In this report we describe different ways to implement SOM on parallel computers. We study the design of massively parallel computers, especially computers with simple processing elements, used for SOM calculations.

It is found that SOM (like many other ANN models) demands very little of a parallel computer. If support for broadcast and multiplication is included very good performance can be achieved on otherwise modest hardware.

1.0 INTRODUCTION

The algorithms we study in this report are Kohonen's self organizing maps (SOM) and variants of them. These maps have been used in pattern recognition, especially in speech recognition [27], but also in robotics and automatic control [40, 46] and telecommunication tasks [3, 32]. This study is part of a series of reports [43, 44, 49] that shows how well suited bit-serial SIMD computers are for simulating artificial neural networks.

As an example of bit-serial SIMD computers, REMAP^3 (reconfigurable, embedded, massively parallel processor project) will be used. As the processing elements are reconfigurable it is possible to include different types of support for different kinds of algorithms. For back-propagation [47] and Hopfield networks [18, 19, 20] a bit-serial multiplier has been found to be essential for the performance [44, 49]. For the implementation of Kanerva's SDM model [25] the multiplier was not needed, instead a counter was suggested [43]. In this report we try to recognize architectural principles and components that are essential for the efficient calculation of Kohonen's models.

In the next section we describe the background of SOM. After that, two sections discuss implementation considerations and ways to map SOM onto a computer architecture. Then follows a section where some of the existing parallel implementations are discussed. Finally, we draw some conclusions concerning the task of designing parallel computers for SOM.

2.0 BACKGROUND

An overview of the different models of self organizing maps and the application areas where they have been used can be found in [26, 28, 29, 30, 31]. Below we only restate the basic models and refer to the references above for more details.

2.1 Competitive Learning

In competitive learning [30, 47] the responses from the adaptive nodes (weight vectors) tend to become localized. After appropriate training the nodes specify clusters or codebook vectors that approximate the probability density functions of the input vectors. Algorithm 1 is an example of a competitive learning algorithm. If the spatial relationships of the resulting feature sensitive nodes are not considered we get a zero-order topology map.

Algorithm 1 Competitive learning (zero-order topology).

1. Find the node (or weight vector) w_i closest to input x.
$$\|x(t_k) - w_c(t_k)\| = \min_i \|x(t_k) - w_i(t_k)\|$$
2. Make the winning node closer to input.
Where $i = c$
$$w_i(t_{k+1}) = w_i(t_k) + \alpha(t_k)\,[x(t_k) - w_i(t_k)]$$
otherwise
$$w_i(t_{k+1}) = w_i(t_k)$$
3. Repeat from step 1 while reducing the learning rate α.

2.1.1 Adding Conscience

A problem with the algorithm above is that instead of placing the nodes according to the input point density function $p(x)$ the nodes are placed as $p(x)^{M/(M+2)}$. Having low dimensional input vectors (i.e. small M) there will be a bias towards the low probability regions. DeSieno [6] has found that adding conscience to the competitive learning algorithm will greatly improve the encoding produced by the map. The idea is that the nodes should be conscientious about how many times they have won, compared to other nodes, see Algorithm 1. That is, every node should win the competition approximately the same

number of times. Another way to improve the clustering is to use a higher-order topology map, like the Kohonen model, this is especially true for non-continuous input probability density functions.

Algorithm 2 Competitive learning with conscience (zero-order topology).

1. Find the node (or weight vector) w_i closest to input x using a conscience c_i as offset (C is a scaling factor).
$$\|x(t_k) - w_c(t_k)\| = \min_i (\|x(t_k) - w_i(t_k)\| + Cc_i)$$
2. Make the winning node closer to input.
Where $i = c$
$$w_i(t_{k+1}) = w_i(t_k) + \alpha(t_k)\,[x(t_k) - w_i(t_k)]$$
otherwise
$$w_i(t_{k+1}) = w_i(t_k)$$
3. Repeat from step 1 while reducing the learning rate α, and increasing the conscience of the winning neuron $c_i(t_{k+1}) = c_i(t_k) + 1$

2.2 Kohonen Learning

In the brain there are many areas, such as the visual and somatosensory cortex, which are organized in a way that reflects the organization of the physical signals stimulating the areas i.e. they are topological maps.

Inspired by that, Kohonen has developed a class of artificial neural network (ANN) models which develop these, so called, self organizing maps (SOM), also referred to as topological feature maps (TFM). They are all models with competitive learning and use first, second, or higher order topological maps [26].

SOM may be formed with unsupervised learning, i.e. without any teacher saying what is right or wrong. This type of SOM is referred to as self organizing feature maps (SOFM), see Algorithm 3.

Algorithm 3 The SOFM algorithm (higher-order topology)

1. Find the node (or weight vector) w_i closest to input x.
$$\|x(t_k) - w_c(t_k)\| = \min_i \|x(t_k) - w_i(t_k)\|$$
2. Find the neighbourhood $N_c(t_k)$.
3. Make the nodes in the neighbourhood closer to input.
Where $i \in N_c(t_k)$
$$w_i(t_{k+1}) = w_i(t_k) + \alpha(t_k)\,[x(t_k) - w_i(t_k)]$$
otherwise
$$w_i(t_{k+1}) = w_i(t_k)$$
4. Repeat from step 1 with ever decreasing neighbourhood N_c and gain sequence α ($0 < \alpha < 1$).

If the resulting maps are to be used as classifiers, and the training example classes are known (i.e. supervised learning), a fine tuning of the SOFM model called learning vector quantization (LVQ) model has been suggested [29, 30]. In the simplest version the difference is that, in Step 3 in Algorithm 3, $\alpha(t_k)$ is negated if w_i belongs to the wrong class. No neighbourhood is used for LVQ.

2.3 Stochastic Competitive Kohonen Learning

Van den Bout and Miller III [52, 53] have suggested a modification to competitive and Kohonen learning which simplifies the calculations by replacing the "analog" signals with stochastic binary signals. In their model, called TInMANN, the mean value of a stochastic binary signal is viewed as an analog signal in the range [0,1]. This signal representation leads to very simple (and space efficient) digital logic for the computations needed in the algorithms. For example, multiplication of stochastic signals can be computed using only a simple AND gate.

By noting that the weight vectors slowly integrate the effects of the environmental stimuli, this can be done by incrementing/decrementing the weights stochastically with a probability proportional to the strength of the input vector. Obviously, more iterations are needed, but as each step is computationally very simple the overall computation time will decrease, see section 5.8.

3.0 IMPLEMENTATION CONSIDERATIONS

SOM models have been implemented on a large number of different parallel computers: Warp [39], Transputers [17, 45, 48], Connection Machine [45], MasPar [11] and in special hardware [8, 24, 37, 38, 52, 53].

Aspects that have to be considered when implementing SOM on parallel computers are: communication facilities, computational capabilities, mapping and partitioning of the algorithm. Some architectures benefit a great deal if floating point numbers can be avoided, and the integer precision needed for the weights should be analysed.

The algorithms in section 2.0 can be divided into four steps:

1. Finding the distance from the input to the nodes.
2. Finding the node closest to input.
3. Determining the neighbourhood to the closest node.
4. Updating the neighbourhood nodes.

The rest of this section will discuss the different aspects of implementing SOM based on these four steps.

3.1 Communication

The weight vectors can be assumed to be distributed over the processing elements (PEs). Thus the first step requires the input vector to be distributed. The most effective way to do this is by utilizing broadcast. If the architecture does not have broadcast, for example if Transputers are going to be used, a nearest neighbour communication like ring or mesh must be used.

For the second step a minimum distance must be found. A minimum could be found by doing N–1 comparisons among the N nodes. Some "global" or "token-ring" communication is needed if the nodes are distributed over many PEs. As seen in section 4.4 a very fast bit-serial method exists to find min/max among PEs if a global-or function exists.

The activation/selection of the neighbours can either be solved as a spatial distance calculation or as a nearest neighbour communication. The time for communication will depend on the neighbourhood size. When the neighbourhood size is small the communication method can be faster than spatial distance calculation. The topology of the computer should support the communication needed, which depends on the spatial topology of the SOM model.

If the input vector (or the difference between the node and input) is stored no communication is needed during step 3.

3.1.1 Summary on Communication

For the basic SOM models broadcast is the most efficient way of communication.

By computing which nodes to be considered as neighbours, instead of using nearest neighbour communication, the computer topology becomes irrelevant for the algorithm. It also makes the time for step 3 constant with respect to the neighbourhood size.

3.2 Computations

For all of the algorithms in section 2.0 the distance is calculated using a Euclidean metric. Another much used metric is the dot-product metric. Both metrics are discussed in this section.

Some model parameters used in this and following sections are:

- N The number of nodes
- N_c Size of the neighbourhood (may depend on time)
- M The dimension of the input vector
- δ Bits used for weights
- n The number of processing elements available.

3.2.1 Euclidean Metric

Using a Euclidean distance metric in the first step means that the distance $\|x(t_k) - w_i(t_k)\|$ is calculated as

$$\sqrt{\sum_j (x_j(t_k) - w_{ij}(t_k))^2}$$

Note that the square-root function does not need to be evaluated, as the square-root is a monotonic function and the result is used for comparison only.

We get the following estimate on the number of computations needed in each step:

1. We must do NM subtractions, NM squarings and NM additions. Note that the result of the subtraction can be reused in step 3 if there is enough memory to store the result.
2. The number of calculations needed for finding minimum is N–1 comparisons (subtractions).
3. If we instead of communicating, calculate the neighbourhood, it should be possible to calculate the spatial distance in less than 3 operations (subtract, square, add) per spatial dimension (usually 1-3).
4. For the updating part we must carry out (N_cM subtractions,)N_cM multiplications andN_cM additions. The subtraction can be carried out in step 1 if there is enough memory to store the result.

The total number of operations is (assuming two spatial dimensions and recalculation of the subtraction)

$$O = 3MN + (N-1) + 6 + 3MN_c.$$

Using a reasonable value forN_c like$N/4$ we get approximately

$$O \approx (1 + 3.75M)\, N \text{ operations per training example.}$$

3.2.2 Dot-Product Metric

By normalizing the input vector (i.e. keeping $\|x(t_k)\| = 1$), and keeping the nodes normalized after updating, the dot-product can be used instead of sum of squares. The distance calculation can then be calculated as a “weight matrix times an input vector” like in many other neural network models.

The matching law is modified to

$$x(t_k)^T w_c(t_k) = \max_i \{ x(t_k)^T w_i(t_k) \}$$

The update law must then be modified to

$$\text{where } i \in N_c(t_k)$$
$$w_i(t_{k+1}) = \frac{w_i(t_k) + \alpha'(t_k)\, x(t_k)}{\left\| w_i(t_k) + \alpha'(t_k)\, x(t_k) \right\|}$$
$$\text{otherwise}$$
$$w_i(t_{k+1}) = w_i(t_k)$$

Stillα' is a monotonically decreasing function, but now $0 < \alpha' < \infty$. The neighbourhood is the same decreasingN_c as before.

We get the following estimate on the number of computations needed in each step:

1. For the dot-product metric the NM subtraction, used by Euclidean distance metric, can be avoided. The squarings are replaced by multiplications.
2. Same as Euclidean metric, but we want to find maximum instead of minimum.
3. Same as Euclidean metric.
4. Using dot-product distance metric we must doN_cM multiplications andN_cM additions for the nominator. For the denominator we must use an additionalN_cM squarings andN_cM additions. We must also do one division.

The total number of operations is (assuming two spatial dimensions)

$$O = 2MN + (N-1) + 6 + 4MN_c + 1.$$

Using a reasonable value forN_c like$N/4$ we get approximately

$$O \approx (1 + 4M)\, N \text{ operations per training example.}$$

The dot-product calculation can in some technologies be very fast (e.g. optical transmission filters). But for our purpose the overhead for the normalization is too time consuming. This will be accentuated if SIMD computers are to be used.

3.2.3 Precision Used for the Weights

If we intend to use a bit-serial architecture, where each PE only operates on one bit at a time, the precision becomes very important. An analysis made by J. Mann [37] shows that at least 8 bits seems necessary for the weights. He also finds that Euclidean distance measures are not as sensitive as dot-product metric to weight precision.

It also seems that the updating rule could be changed to an increment/decrement of the weight, depending on the sign of the difference between the weight and the input without much influence on the performance of the SOM algorithm.

Hammerstrom and Nguyen have found the SOM to be sensitive to error bias from bit truncation, more sensitive than to average quantization error from reduced precision. They also reduce the error accumulation using saturation arithmetic (on overflow the max/min value is used).

3.2.4 Summary and Comments on Computation

Almost half of the operations computed are multiplications, and therefore support for multiplication is very important. As

calculations using short fixed-point numbers seem possible, bit-serial computers become interesting. The Euclidean metric uses less operations for its computation, and is less sensitive to weight quantization, and is therefore more attractive to use than dot-product.

Many have used CUPS (connection updates per second) as a measure of performance on SOM algorithms. It has been used as the number of connections *MN* multiplied by the number of updates per second *u* i.e *MNu*. This despite the fact that very few connections are actually updated when using small neighbourhoods.

Even if the word *updates* is misleading, the CUPS measure, used in relation to the FLOPS (floating point operations per second) or IPS (instructions per second) measures, can be used as an indication of the efficiency of the architecture (on SOM algorithms). We may define an efficiency measure like:

$$E = \frac{\text{operations per second}}{\text{maximum number of operations}} = \frac{Ou}{\text{IPS}_{\max}} \approx \left(\frac{3.75MNu}{\text{IPS}_{\max}} = 3.75\frac{\text{CUPS}}{\text{IPS}_{\max}} \right) \quad \textbf{(EQ 1)}$$

Where $\text{IPS}_{\max}$ is the maximum number of operations per second achievable on the computer. The corresponding measure for floating point calculations is:

$$E = 375\frac{\text{CUPS}}{\text{FLOPS}_{\max}}$$

4.0 MAPPING SOM ONTO A COMPUTER ARCHITECTURE

4.1 Computer Architectures

One of the most used divisions of architectures is due to Flynn [10]. He divided the computers into groups according to the number of instruction streams and the number of data streams:

1. SISD - Single Instruction stream, Single Data stream.
2. SIMD - Single Instruction stream, Multiple Data streams.
3. MISD - Multiple Instruction streams, Single Data stream. (Anomaly of the division)
4. MIMD - Multiple Instruction streams, Multiple Data streams.

Single instruction stream multiple data stream (SIMD) computers is a computer model where the same instruction is executed in each of the processing elements (PEs). Using this model it is possible to achieve massive parallelism at small cost. It is found that almost all ANN algorithms fit very well into the SIMD model [44]. The SOM algorithms in section 2.0 also seem to map easily onto the SIMD computer model. This hypothesis will be found to be true later in this section. Below there follows a discussion on how to implement SOM on a SIMD computer.

4.2 Degree of Parallelism

Looking at an ANN algorithm such as SOM there are (at least) six different ways of achieving parallelism [44]. The typical degree of parallelism varies widely between the six different kinds, as the table below shows.

Parallelism	Typical range SOM:
Training session	$10 - 10^3$
Training example	$10 - 10^7$
Forward-Backward	1 - 2
Node (neuron)	$100 - 10^6$
Weight (synapse)	$2 - 10^4$
Bit	1 - 64

To utilize the computing resources of a massively parallel computer (thousands of processing elements) efficiently the table indicates that we must use at least one of the following dimensions:

Training session parallelism.
Training example parallelism.
Node parallelism.
Weight parallelism.

Note that the first two dimensions of parallelism are of interest only in batch processing situations, i.e. when training the network. If the network is to be to used in a real-time situation, interacting with the outside world, training session and training example parallelism would be unavailable. In those cases, node and/or weight parallelism must be chosen, possibly in combination with e.g. bit and layer parallelism.

4.3 Node Parallelism on SIMD Computers

Unfortunately the amount of node parallelism used for updating the nodes varies with the size of neighbourhood. Still the node parallelism is the most used and maybe the most natural mapping for SIMD computers. The mapping may be visualized for a 1D map topology on a linear processor array as in Figure 1.

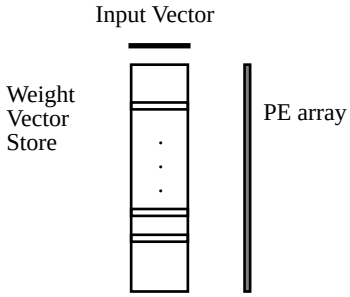

Figure 1 Using node parallelism to map SOM onto a linear processor array.

For the four steps we then get the following estimate of the number of computation steps needed, using Euclidean distance metric and having the same number of processing elements as the number of nodes, i.e. $n = N$.

1. Calculating the distance requires *M* subtraction steps, *M* squaring steps and *M* addition steps. The result of the subtraction can be reused in step 4 if there is enough memory to store the result.
2. The number of calculations needed for finding the minimum distance varies with the communication structure. Using bit-serial arithmetic and global-or function, as later described in section 4.4, the comparison can be computed in less than $2(2\delta + \log_2 M)$ (Two times the number of bits in the sum of squares) steps.

3. The time to determine the neighbourhood varies with the communication structure, but it should always be possible to calculate in less than 3 operations per spatial dimension.
4. Updating the weights requires (M subtractions), M multiplications and M additions. Note that the efficiency during this step is only N_c / N i.e. the efficiency will go down as the neighbourhood shrinks.

The total number of steps is, using a 2D map as an example, approximately

$$O = 3M + 2(2\delta + \log_2 M) + 6 + 3M$$

4.4 Bit-serial SIMD Implementation

The majority of massively parallel processors use bit-serial arithmetic, and that is also the basic mode of operation for our own research machine, REMAP[3]. Therefore, we would like to analyse the algorithms down to bit-level. For the majority of operations using bit-serial PEs, the processing times grow linearly with the data length used. E.g. the time to do a bit-serial addition is the same time as to read the operands and store the result (3δ cycles). This may be regarded as a serious disadvantage (e.g when using 32- or 64-bit floating point numbers), or as an attractive feature (use of low precision data speeds up the computations accordingly). In any case, bit-serial data paths simplify communication in massively parallel computers.

As concluded in section 3.2.4 support for multiplication is important. Unfortunately, very few bit-serial computers have support for bit-serial multiplication, and without such, multiplication time grows quadratically with the data length. However, with an inclusion of a bit-serial multiplier [9, 44, 55] the multiplication of two δ bit numbers can be performed in 4δ cycles (i.e. the time to read the operands and store the result).

Using the natural mapping for SIMD computers (node parallelism) and having $n = N$ we get the following number of cycles for the SOM algorithm using Euclidean metric.

1. We have to do $3\delta M$ cycles for subtraction, $4\delta M$ cycles for squaring and $6\delta M$ cycles for addition. If the result from the multiplier is used directly for addition (i.e. without storing the result in memory) the total time for squaring and adding will be $8\delta M$.
2. Finding minimum (or maximum) using bit-serial working mode can be implemented very efficiently. A global-or function is needed to check a bit "slice" if all bits are equal, also the means to turn off PEs depending on the result of the previous operation are needed. Assuming we want to find minimum (i.e. using Euclidean distance metric), the search starts by examining the most significant bit of each value. If anyone has a zero, all PEs with a one are turned off (otherwise we restore the previous state). The search goes on in the next position, and so on, until all bit positions have been treated. The time for this search is independent of the number of values compared, it depends only on the data length. The maximal number of cycles needed will be two times the length of the data i.e $2(2\delta + \log_2 M)$. To resolve multiple minimums a select first network [9] can be used to select the first active processor.
3. Wanting to have constant time for this step we want to calculate the neighbourhood (instead of communicate it). There are a number of variants depending on the (spatial) distance measure and the map topology. If 7 bits are used for the spatial coordinates, the following estimates for the cycle count can be given:

- Euclidean (spatial) distance

  A second order topology map can be computed in 206 cycles

- City block (spatial) distance

  A first order topology map can be computed in 48 cycles

  A second order topology map can be computed in 96 cycles

4. Updating of the weight vectors can be computed with $3\delta M$ cycles for subtraction, $4\delta M$ cycles for multiplication and $3\delta M$ cycles for addition. If the global constant α is loaded in parallel from the controller, and the result from the multiplier is used directly for the addition, the number of cycles in step 4 will be $7\delta M$.

There are of course some cycles needed for overflow tests and initiations but the above indicates that SOM could be calculated in typically: $18\delta M + 250$ cycles. Note, that we assume $n = N$. Using the approximate total number of operation $O \approx (1 + 3.75M)N$, we can calculate the efficiency $E = (Ou) / \mathrm{IPS}_{\max}$.

Let C stand for the clock frequency. As the time for multiplication is 4δ the $\mathrm{IPS}_{\max}$ is $(Cn) / (4\delta)$. The update rate will be $u = (Cn) / (18\delta MN + 250N)$ and the efficiency

$$E = \frac{4(1 + 3.75M)\delta}{18\delta M + 250}$$

The asymptotic efficiency will be 83%. Already for M=10 and δ=8 the efficiency is 73%. These figures show that a bit-serial SIMD computer can be used very efficiently for SOM calculations.

4.5 Other Forms of Parallelism

If the input vectors are long (i.e. large M) and the neighbourhood N_c is small, a "transposed" mapping could be considered. This is the same as the columnwise mapping suggested for SDM [43]. For SOM this would correspond to weight parallelism with iteration over the nodes, see Figure 2.

Using this mapping the summation of squares must be done across the PEs, and special hardware (e.g. an adder-tree) should be included for efficiency reasons. However, with this hardware available it is possible to pipeline the addition and comparison (carried out in the controller) with the subtraction and squaring (carried out in the processor array).

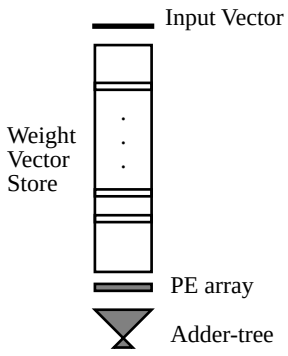

Figure 2 Using weight parallelism to map SOM onto a linear processor

For the last step, weight parallelism is faster than node parallelism with a factor M/N_c. This will be a considerable factor when N_c is small and M is large. The second and third steps are almost "for free" for weight parallelism as pipelining can be used. As the number of nodes N often is larger than the number of elements M in the input vector, and for weight parallelism we only have $n = M$ PEs, this mapping will be less efficient than node parallelism during the first step. The factor is $(3M)/(2N)$.

It seems that the most efficient mapping during the first step is node parallelism and during the fourth step weight parallelism (as for the SDM model [43]). But we have not found any way to achieve mixed mapping for SOM onto any normal SIMD computer.

Note that if $2N > 5M$, no matter how fast the three last steps are when weight parallelism is used, it will still be less efficient than the node parallel version. This is because its first step takes longer than the total time of the node parallel version.

5.0 IMPLEMENTATIONS ON PARALLEL COMPUTERS

In the following subsections many of the parallel computers used for SOM are described. Performance figures for the SOM algorithm on these computers are also given if they are available.

A summary of the discussed implementations is shown in Figure 3.

5.1 CNAPS

CNAPS (Connected Network of Adaptive ProcessorS) manufactured by Adaptive Solutions is one of the first architectures developed especially for ANN. It was called X1 in the first description by Hammerstrom [14]. It is a 256 PE SIMD machine with a broadcast interconnection scheme. Each PE has a multiply (9x16bit) and add arithmetic unit and a logic-shifter unit. It has 32 general (16bit) registers and a 4kByte weight memory. There are 64PEs in one chip. 1, 8 or 16 bits can be used for weights and 8 or 16 bits for activation.

The performance of CNAPS on SOFM was reported by Hammerstrom and Nguyen in [15]. The figures are based on a 20MHz version of CNAPS. Best match using Euclidean distance measure, having N=512 nodes and M=256 elements per vector (δ=16), can be carried out in 215 µs. Making their CUPS figure comparable to others the performance will be about 183 MCUPS. A CNAPS computer can maximally achieve 10240 MIPS on dot-product operations. The efficiency is thus:

$$E = \frac{3.75\,(183)}{10240} = 7\%$$

More figures are needed to be able to analyse where the bottleneck is. The low efficiency may depend on the high maximal performance, achieved in an operation mode which can not be utilized for SOM.

5.2 Connection Machine

The Connection Machine [16, 50] manufactured by Thinking Machines Corporation (TMC) is for the moment the most massively parallel machine built (from 8k up to 64k processing elements). In addition to its large number of processors, two of its strong points are its powerful hypercube connection for general communication and the multidimensional mesh connection for problems with regular array communication demands. In the CM-2 model TMC also added floating point support, implemented as one floating point unit per 32 PEs. This means 2048 floating point units on a 64k machine, giving a peak performance of 10 GFlops. CM-2 is one of the most popular parallel computers for implementing ANN algorithms.

Obermayer et al. have implemented large SOM on the CM-2 [45]. They used node parallelism and up to 16k PEs (nodes). The input vector length M was varied and lengths of up to M=900 were tested. 48 MCUPS were achieved on a problem with n=N=16384 and M=100. As they used a bell-shaped Gaussian function for neighbourhood calculations, an efficiency measure according to equation (EQ 1) would be meaningless.

The same authors have also implemented the same algorithm on a self built computer with 60 T800 (Transputer) nodes connected in a systolic ring. Besides algorithmic analysis they have benchmarked the two architectures. Having M=100, n=30 and N=14400 they achieved 2.4 MCUPS.

The conclusion is that the CM-2 (16k PEs) with floating point support is equal to 510 Transputer nodes for the SOM. As a 16k CM-2 has 512 Weitek floating point units, each with approximately the same performance as one T800 on floating point calculations, it can be concluded that SOM basically is computation bound. In a "high-communication" variant of SOM where broadcast could not be used efficiently a 30 node Transputer machine would run at one third of the CM-2 speed.

5.3 L-Neuro

The Laboratoires d'Electronique Philips (LEP), Paris, have designed a VLSI chip called L-Neuro. It contains 16 processors working in SIMD fashion. In association with these chips Transputers are imagined as control and communication processors. The chip has support for multiplications with a multiply step. Weights are represented by 2-complement numbers over 8 or 16 bits, and the states of the neurons are coded over 1 to 8 bits.

Duranton and Sirat [7, 8] have described implementations of both SOM, Hopfield and BP networks on this architecture. However, no figures of performance were given.

5.4 MasPar

MasPar MP-1 [1, 5, 42] is a SIMD machine with both mesh and global interconnection style of communication. It has floating point support, both VAX and IEEE standards. The number of processing elements can vary between 1024 and 16384. Each PE has 40 32-bit registers, a 4-bit integer ALU, floating point "units" for Mantissa and Exponent, addressing unit for local address modifications, and a 4-bit broadcast bus. MP-1 has a peak performance, for a 16k PE machine, of 1500 MFlops single-precision [42].

In [4, 12, 13] Grajski, Chin et al. have implemented BP and SOM. The mapping of SOM into MP-1 uses node parallelism. It was measured to give 17.2 MCUPS on a 4k machine when 16-dimensional input vectors were used. They report that high efficiency is achieved using the MP-1 and that the performance figures increase with the dimensionality (up to 18 MCUPS).

5.5 REMAP^3

The goal of REMAP^3 is to construct modules for ANN computation. A typical module for SOM would consist of a few thousand bit-serial processors. If the processing elements needed for ANN were integrated on a VLSI chip more than 128 PEs per chip would be possible. On a single board, 1k processors with memory would be possible. If implemented on VLSI a clock frequency above 20 MHz would be of no problem. This makes it possible to achieve an update rate of approximately 539 per second on an $n = N = 2048$ problem (M=128 andδ = 16) and more than 11800 updates per second on the smaller problem ($n = N = 1024$, $M = 10$ andδ = 8). The corresponding CUPS (and efficiency) figures would be 141 MCUPS (=> E=83%) and 121 MCUPS (=> E=71%), respectively.

5.6 Transputer

The Transputer [22, 54] is a single chip 32-bit microprocessor. It has support for concurrent processing in hardware which closely corresponds to the Occam [2, 21, 23] programming model. It contains onchip RAM and four bi-directional 20 Mbits/sec communication links. By wiring these links together a number of topologies can be realized. Each Transputer of the T800 type is capable of 1.5 MFlops (20MHz) and architectures with up to 4000 Transputers are being built [54].

The SOM model has been implemented on Transputers by Hodges et al. [17], Siemon and Ultsch [48] and Obermayer et al. [45]. All three implementations distribute the nodes over the PEs and use ring communication to distribute the input vector and to find the global minimum. As long as the neighbourhood is larger than the number of PEs this mapping is quite efficient. Good performance will also be achieved for high input dimension.

Hodges et al. presented an equation for the performance but no concrete implementation.

Siemon and Ultsch state a performance of 2.7 MCUPS on a 16 Transputer machine. Having N=128x128, M=17, u=25000/2546 the total number of operations would be $O = 991000$. Each Transputer can give 1.5 MFlops so the maximum number of Flops would be 24MFlops. Then the efficiency would be E = 42%

Obermayer et.al have implemented SOM on, and compared the performance of, Connection Machine and Transputers, see section 5.2.

A more general implementation framework, called CARELIA, has been developed by Koikkalainen and Oja [35]. The neural network models are specified in a CSP-like formalism [33, 34, 35]. The simulator is currently running on a network of Transputers and one of the models implemented is SOM. The performance of the simulator has not been reported.

5.7 Warp

Warp is a one-dimensional array of 10 or more powerful processing elements (PEs) developed at Carnegie-Mellon University in 1984-87 [36]. Each cell/PE has a microprogrammable controller, a 5 MFlops floating-point multiplier, a 5 MFlops floating-point adder and a local memory. Communication between adjacent cells may be conducted in parallel over two independent channels: a left-to-right X channel and bidirectional Y channel.

An implementation of SOM on Warp has been described by R. Mann and Haykin [39]. When they used training example parallelism between 6 and 12.5 MCUPS were achieved. Because of a fixed communication overhead (0.01s) at the start of each batch, better performance could not be achieved. Some minor problem with the topology ordering process when using training example parallelism were reported. They suggested that either the map starts at some order instead of at random state, or that the map is trained sequentially for the first 100-1000 steps, after which the training example parallelism is “turned on”.

The Warp has 10 PEs, each having 10 MFlops, giving it a total of 100MFlops. Having N=1024 and M=128 the implementation could run at u=5.3x18 updates per second (each batch (epoch) was 18 training examples), giving it 12.5 MCUPS and an efficiency of E=47%.

5.8 TInMANN

Van den Bout et al. [41, 52, 53] have suggested a stochastic all digital implementation of Competitive Kohonen learning (see section 2.3) called TInMANN.

Their modifications to the SOFM model make it possible to use very simple PEs. A typical node would consist of two registers (10-12 bits), two adders or subtractors, two flags, memory for weights, gated broadcast, global-or function and some control logic. A VLSI version of TInMANN has been implemented where each node used 4000 transistors [41]. Using 10-bit weights the memory could be used for three to four weights per node, i.e. the input vector length M is very limited. The update rate, using 20MHz clock, is 267000 updates per second per node using $M = 3$. If one million transistors were used a 250 node chip could be constructed and thus giving 200 MCUPS per chip. As broadcast is used there is no problem to extend the system outside the chip boundaries.

A “rapid prototyping” version of the architecture in reconfigurable logic (XILINX) is also reported [53]. To eliminate some complexity and space, the architecture uses bit-serial nodes. Each chip (X3020) contains 3 PEs and external RAM is used for weights (i.e. much larger M can be used). One circuit board, called Anyboard, contains up to 8 X3020 and thus contains up to 21 PEs. This project was not completed. However, simple calculations showed that the interface between the host and the bit-serial nodes over the IBM PC bus was the major bottleneck. Ignoring the bus interface, the nodes were quite fast [51].

COMPUTER	n	Max MIPS (MFLOPS)	N	M	δ	u	MCUPS	E
CM	16384	2500	16384	100	FP		48	-%
MasPar	4096	376	4096	16	FP	250	16	16%
	4096	376	4096	256	FP	17	18	18%
Warp	10	100	1024	16	FP	522	8.6	32%
	10	100	1024	128	FP	95.4	12.5	47%
Transputer								
Obermayer	30	45	14400	100	FP		2.4	-%
Siemon	16	24	16384	17	FP	9.8	2.7	40%
CNAPS	256	10240	512	128	8		210	8%
	256	10240	512	128	16		180	7%
REMAP[3]	128	80	2048	128	8	67	17.5	82%
	1024	640	1024	10	8	11800	121	71%
	2048	1280	2048	128	8	1070	280	82%
	2048	640	2048	128	16	539	141	83%
TInMANN	250	-	250	3	10	267000	200	-%

Figure 3 The performance figures on SOFM simulations, for all the discussed computers. The efficiency *E* is calculated as 3.75 CUPS/IPS_{max}. No efficiency figure has been given if the CUPS figure is incompatible with the other figures.The number of processors is denoted by *n*, the number of nodes by *N*, the dimension of the input vector by *M*, the number of bits used for weights byδ and the update rate by *u*.
Note that if the computer manufacturer has been given “to high” maximum performance figures, the efficiency figure will be proportionally lower.

6.0 CONCLUSION

Unless especially tuned for SOM as REMAP[3], none of the computers studied have very high efficiency. As many of the performance figures given in the literature are measured under vastly different experimental setups, the figures given for MCUPS and efficiency, should be used cautiously. Still, the figures together with our analysis, indicate that the communication structure and the control mechanism are of little importance for the calculation of SOM. When designing a high performance SOM computer the design effort must be on implementing efficient (with respect to time and area) multipliers which can be supported with operands from the controller, e.g. using broadcast, together with high bandwidth access to local memory.

The fact that the neighbourhoodN_c becomes very small towards the end of the training session seems easier to take advantage of on a coarse grain MIMD computer. Even if only one PE is working there will be relatively fewer idle processors during the updating step, thus giving the MIMD computer a possibly better efficiency.

The analysis also indicates that SOM can be mapped efficiently onto bit-serial SIMD computers. The only requirement is that a bit-serial multiplier is included to support the many multiplications.

By modifying the algorithm to include a “stochastic signal” it is possible to run competitive learning without using multiplication. This makes it possible to achieve enormous update rates, but as the updates are of a different kind, it is difficult to compare it to the original method with respect to speed. Still, this modification is very interesting as an alternative to ordinary competitive learning algorithms, as it reduces the architectural components needed for the computation. Moreover, bit-serial SIMD computers could be considered as one of the prime candidates for an efficient implementation of this modified SOM algorithm.

The natural and totally dominating dimension of parallelism is the node parallelism. Training example parallelism has been used, but it seems that only relatively small batch sizes may be used as there will otherwise be failures in the topological ordering. The weight parallelism and the mixed mapping, which were successfully used in the computation of Kanerva’s SDM model [43], can not be used efficiently for the calculation of the SOM model. This is due to the fact that the adaptation is taking place in the weight matrix which is also used for the selection phase, whereas for SDM the adaptation is made in a separate matrix.

7.0 REFERENCES

[1] Blank, T. "The MasPar MP-1 Architecture." In *Proceedings of COMPCON Spring 90*, pp. 20-24, San Francisco, CA, 1990.

[2] Bowler, K. C., et al. *An Introduction to OCCAM 2 Programming*. Chartwell-Bratt. 1987.

[3] Bradburn, D. S. "Reducing transmission error effects using a self-organizing network." In *International Joint Conference on Neural Networks*, Vol. 2, pp. 531-537, Washington, DC, 1989.

[4] Chinn, G., et al. "Systolic array implementations of neural nets on the maspar MP-1 massively parallel processor." In *International Joint Conference on Neural Networks*, Vol. 2, pp. 169-173, San Diego, 1990.

[5] Christy, P. "Software to support massively parallel computing on the MasPar MP-1." In *Proceedings of COMPCON Spring 90*, pp. 29-33, San Fransisco, CA, 1990.

[6] DeSieno, D. "Adding a conscience to competitive learning." In *International Conference on Neural Networks*, Vol. 1, pp. 117-124, San Diego, 1988.

[7] Duranton, M. and J. A. Sirat. "Learning on VLSI: A general purpose digital neurochip." In *International Conference on Neural Networks*, Washingtin, DC, 1989.

[8] Duranton, M. and J. A. Sirat. "Learning on VLSI: A gerneral-purpose digital neurochip." *Philips Journal of Research*. Vol. 45(1): pp. 1-17, 1990.

[9] Fernström, C., I. Kruzela and B. Svensson. *LUCAS Associative Array Processor - Design, Programming and Application Studies*. Vol 216 of *Lecture Notes in Computer Science*. Springer Verlag. Berlin. 1986.

[10] Flynn, M. J. "Some computer organizations and their effectiveness." *IEEE Transaction on Computers*. Vol. C-21: pp. 948-60, 1972.

[11] Grajski, K. A. "Neurocomputing using the MasPar MP-1." (Technical Report No. 90-010), Ford Aerospace, 1990.

[12] Grajski, K. A. "Neurocomputing using the MasPar MP-1." *Digital Parallel Implementations of Neural Networks.* Przytula and Prasanna ed. Prentice-Hall. (Forthcoming). 1992.

[13] Grajski, K. A., et al. "Neural Network Simulation on the MasPar MP-1 Massively Parallel Processor." In *The International Neural Network Conference*, Paris, France, 1990.

[14] Hammerstrom, D. "A VLSI architecture for high-performance, low-cost, on-chip learning." In *International joint conference on neural networks*, Vol. 2, pp. 537-543, San Diego, 1990.

[15] Hammerstrom, D. and N. Nguyen. "An implementation of Kohonen's self-organizing map on the Adaptive Solutions neurocomputer." In *International Conference on Artificial Neural Networks*, Vol. 1, pp. 715-720, Helsinki, Finland, 1991.

[16] Hillis, W. D. and G. L. J. Steel. "Data parallel algorithms." *Communications of the ACM*. Vol. 29(12): pp. 1170-1183, 1986.

[17] Hodges, R. E., C.-H. Wu and C.-J. Wang. "Parallelizing the self-organizing feature map on multi-processor systems." In *International Joint Conference on Neural Networks*, Vol. 2, pp. 141-144, Washington, DC, 1990.

[18] Hopfield, J. J. "Neural networks and physical systems with emergent collective computational abilities". *Proceedings of the National Academy of Science USA*. 79: pp. 2554-2558, 1982.

[19] Hopfield, J. J. "Neurons with graded response have collective computational properties like those of two-state neurons". *Proceedings of the National Academy of Science USA*. 81: pp. 3088-3092, 1984.

[20] Hopfield, J. J. and D. Tank. "Computing with neural circuits: A model." *Science*. Vol. 233: pp. 624-633, 1986.

[21] INMOS Limited. *Occam programming model*. Prentice-Hall. 1984.

[22] INMOS Limited. "The Trasputer family 1987". 1987.

[23] INMOS Limited. *Occam 2 Reference Manual*. Prentice-Hall. London. 1988.

[24] Johnson, M. J., N. M. Allinson and K. J. Moon. "Digital realisation of self-organising maps." In *Neural Information Processing Systems 1*, pp. 728-738, Denver, CO, 1988.

[25] Kanerva, P. *Sparse Distributed Memory.* MIT press. Cambridge, MA. 1988.

[26] Kangas, J. A., T. K. Kohonen and J. T. Laaksonen. "Variants of self-organizing maps." *IEEE Transaction on Neural Networks*. Vol. 1(1): pp. 93-99, 1990.

[27] Kohonen, T. "The 'neural' phonetic typewriter." *Computer*. Vol. 21(3): pp. 11-22, 1988.

[28] Kohonen, T. *Self-Organization and Associative Memory.* (2nd ed.) Springer-Verlag. Berlin. 1988.

[29] Kohonen, T. "Improved versions of learning vector quantization." In *International Joint Conference on Neural Networks*, Vol. 1, pp. 545-550, San Diego, 1990.

[30] Kohonen, T. "The self-organizing map." *Proceedings of the IEEE*. Vol. 78(9): pp. 1464-1480, 1990.

[31] Kohonen, T. "Some practical aspects of the self-organizing maps." In *International Joint Conference on Neural Networks*, Vol. 2, pp. 253-256, Washington, DC, 1990.

[32] Kohonen, T., et al. "An adaptive discrete-signal detector based on self-organizing maps." In *International Joint Conference on Neural Networks*, Vol. 2, pp. 249-252, Washington, DC, 1990.

[33] Koikkalainen, P. "MIND: a specification formalism for neural networks." In *International Conference on Artificial Neural Networks*, Vol. 1, pp. 579-584, Helsinki, Finland, 1991.

[34] Koikkalainen, P. and E. Oja. "Specification and implementation environment for neural networks using communication sequential processes." In *International Conference on Neural Networks*, San Diego, CA, 1988.

[35] Koikkalainen, P. and E. Oja. "The CARELIA simulator: a development and specification environment for neural networks." (Research Report No. 15/1989), Lappeenranta Univ. of Tech, Finland, 1989.

[36] Kung, H. T. "The Warp computer: architecture,implementation and performace." *IEEE Transaction on Computers*. Vol. Dec: 1987.

[37] Mann, J. "The effects of circuit integration on a feature map vector quantizer." In *Neural Information Processing Systems 2*, pp. 226-231, Denver, CO, 1989.

[38] Mann, J. and S. Gilbert. "An analog self-organizing neural network chip." In *Neural Information Processing Systems 1*, pp. 739-747, Denver, CO, 1988.

[39] Mann, R. and S. Haykin. "A parallel implementation of Kohonen feature maps on the Warp systolic Computer." In *International Joint Conference on Neural Networks*, Vol. 2, pp. 84-87, Washington, DC, 1990.

[40] Martinetz, T. M., H. J. Ritter and K. J. Schulten. "Three-dimensional neural net for learning visuomotor coordination of robot arm." *Transaction on neural networks*. Vol. 1(1): pp. 131-136, 1990.

[41] Melton, M., et al. "VLSI Implementation of TInMANN." In *Advances in Neural Information Processing Systems 3*, Denver, CO, 1990.

[42] Nickolls, J. R. "The design of the MasPar MP-1: a cost effective massively parallel computer." In *Proceedings of COMPCON Spring 90*, pp. 25-28, San Fransisco, CA, 1990.

[43] Nordström, T. "Sparse distributed memory simulation on REMAP3." (Research Report No. TULEA 1991:16), Luleå University of Technology, Sweden, 1991.

[44] Nordström, T. and B. Svensson. "Using and designing massively parallel computers for artificial neural networks." (Research Report No. TULEA 1991:13), Luleå University of Technology, Sweden, 1991.

[45] Obermayer, K., H. Ritter and K. Schulten. "Large-scale simulations of self-organizing neural networks on parallel computers: application to biological modelling." *Parallel Computing*. Vol. 14(3): pp. 381-404, 1990.

[46] Ritter, H. J., T. M. Martinetz and K. J. Schulten. "Topology conserving maps for learning visuo-motor-coordination." *Neural Networks*. Vol. 2(3): pp. 159-168, 1989.

[47] Rumelhart, D. E. and J. L. McClelland. *Parallel Distributed Processing; Explorations in the Microstructure of Cognition.* Vol I and II MIT Press. Cambridge. 1986.

[48] Siemon, H. P. and A. Ultsch. "Kohonen networks on transputers: Implementation and animation." In *International Neural Network Conference*, Vol. 2, pp. 643-646, Paris, 1990.

[49] Svensson, B. and T. Nordström. "Execution of neural network algorithms on an array of bit-serial processors." In *10th International Conference on Pattern Recognition, Computer Architectures for Vision and Pattern Recognition*, Vol. II, pp. 501-505, Atlantic City, New Jersey, USA, 1990.

[50] Thinking Machines Corporation. "Connection Machine, Model CM-2 Technical Summary." (Version 5.1), T M C Cambridge, Massachusetts, 1989.

[51] Van den Bout, D. E. 1991. Personal communication.

[52] Van den Bout, D. E. and T. K. M. III. "TInMANN: The integer Markovian artificial neural network." In *International joint conference on neural networks*, Vol. 2, pp. 205-211, Washington, 1989.

[53] Van den Bout, D. E., W. Snyder and T. K. Miller III. "Rapid prototyping for neural networks." *Advanced Neural Computers*. Eckmiller ed. North-Holland. Amsterdam. 1990.

[54] Whitby-Stevens, C. "Transputers — past, present, and future." *IEEE Micro*. (December): pp. 16-82, 1990.

[55] Wilson, S. S. "Neural computing on a one dimesional SIMD array." In *11:th International Joint Conference on Artificial Intelligence*, pp. 206-211, Detroit, Michigan, USA, 1989.