Parallel implementation of the Kohonen algorithm on Transputer

Roberto Togneri, Member I.E.E.E
Yianni Attikiouzel, Member I.E.E.E, Member I.E.E
Department of Electrical and Electronic Engineering
The University of Western Australia
Nedlands 6009, Western Australia

Abstract

In this paper a parallel implementation of the Kohonen algorithm is proposed using partitioning of the network. This allows an exact implementation of the Kohonen algorithm as opposed to partitioning the data. By using a simple routing strategy the parallel Kohonen algorithm was tested on a PC based transputer network without the need for any special distributed operating system. The execution time was measured for different sized networks and number of transputers. The execution time decreased as the number of transputers increased. However, for comparatively small sized neural networks the communication overhead caused the execution time to increase when more transputers were used. Thus, the proposed parallel implementation of the Kohonen algorithm is not suitable for massively parallel architectures.

1 Introduction

The Kohonen self-organising map [1, 2] is a neural network algorithm for vector quantisation, classification and data reduction [3, 4]. A problem with the algorithm is the computational load as the network size increases. A large number of iterations are needed because of the slow convergence of the algorithm and, if extended to higher dimension maps [5], the memory requirements also become significant. These factors severely limit its use on conventional (serial) computers. To alleviate this a parallel implementation of the Kohonen algorithm will allow both faster computations and smaller local memory requirements. One way to achieve this is to partition the data among parallel processors (data partitioning) [6], however, this is not an exact implementation of the Kohonen algorithm and can cause problems. By partitioning the network itself among the parallel processors (network partitioning) the algorithm can be implemented exactly [6]. With network partitioning, however, the communication overhead is increased and can become significant preventing a massively parallel implementation of the Kohonen algorithm.

In this paper a parallel implementation of the algorithm on transputers using network partitioning is described. The effect of varying the number of transputer nodes and the size of the network will demonstrate the efficiency of the parallel implementation.

2 Transputer Farm Architecture

The transputers consisted of one T414 and up to 12 25 MHz T800 transputers all running off an IBM-AT host. The T414 acted as the root transputer connected to the host and was also able to run a graphics monitor. The T800 transputers came on plug-in boards with 4 T800's on each board. The T414 had 2kB of fast on-chip memory while the T800 had 4kB. All transputers had 1 Mb of slower RAM. The fast on-chip memory was exploited by placing the stack and code in this area. Due to power loading two of the boards were run from a separate PC properly

CH 3065-0/91/0000-1717 $1.00 © IEEE

grounded with the host. There was no distributed operating system available and programming was performed using the 3L C compiler. The lack of a distributed OS meant that a simple message routing system had to be developed. The easiest way to do this was to use a farm architecture. In a farm or master/slave setup a master node passes messages to and from each slave node. There is no slave to slave communication. This makes the implementation less topology dependent, the slave nodes only need to know where the master node is. Other than the ability to communicate between transputers there is no need for any specialised distributed operating system.

The hardware configuration initially attempted is shown in Fig. 1. It was later found that more than 10 transputers could not be run and the dual chain configuration of Fig. 2 was used. With this arrangement 12 transputers (6 per chain) could be run satisfactorily.

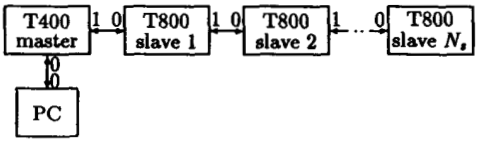

Figure 1: Transputer architecture for single chain configuration. Link connections as shown.

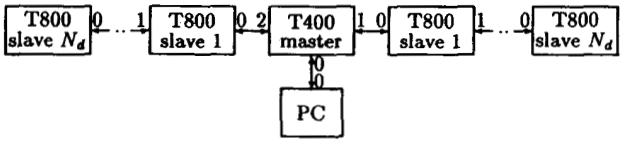

Figure 2: Transputer architecture for dual chain configuration. Link connections as shown.

3 Serial Kohonen Algorithm

The Kohonen self-organising map is a two dimensional topological arrangement of neurons. Associated with each neuron is a weight vector, w_i. The input to the network, a vector x,is connected to each neuron through the weights. The dimension of the weight vector and input vector are the same.

Training the network using the Kohonen algorithm involves cycling through the sequence of input vectors. Each iteration consists of a search part and a weight update part. During the search part each neuron computes the Euclidean distance between its own weight vector and the input vector:

$$dist_i = \|x - w_i\| \tag{1}$$

The winning neuron, win_i, is chosen to be the one which exhibits the smallest Euclidean distance:

$$win_i = \min_i dist_i \tag{2}$$

In the weight update part the neurons which fall within the square topological neighbourhood, Nh_c, of the winning neuron have their weights modified (updated) according to the rule:

$$w_i^{new} = w_i^{old} + \alpha(x - w_i^{old}) \tag{3}$$

where α is the gain factor.

The training of the map is done in two phases, the ordering phase followed by the convergence phase. In the ordering phase the gain factor, α, is linearly decreased from an initial value to zero while the neighbourhood size, Nh_c, is linearly decreased from an initial value to one. In the convergence phase the neighbourhood size, Nh_c, is fixed at 1.

4 Parallel Kohonen Algorithm

The parallel implementation of the Kohonen self-organising algorithm using a master/slave processor arrangement is as follows.

The master program runs on the root T414 and does all the I/O and main program control. The former is of necessity since on a transputer only the root T414 can communicate with the PC host.

1. The master program divides the self-organising grid of neurons of size n_x^g by n_y^g (i.e. number of global neurons $n^g = n_x^g n_y^g$) into equal partitions of size n_x^l by n_y^l (i.e number of local neurons $n^l = n_x^l n_y^l$) and assigns a slave node to each partition (thus there are $N_t = n^g/n^l$ transputer slaves).
2. The input training vector is broadcast to all slave nodes.
3. The slave nodes proceed to find the local winning neuron as given by Eqs. 1 and 2. The winning neuron and its distance measure is returned to the master node.
4. The master node determines the global winning neuron from the N_t local winning neurons. The bounds of the neighbourhood around the winning neuron is determined and broadcast to all slave nodes.
5. Each slave node determines which part, if any, of their own local map falls within the neighbourhood and updates the neurons according to Eq. 3.

In this algorithm the master program does take part in some of the computations. This is normally not desirable since the slave nodes remain idle and a T414, unlike the T800, does not have a floating point unit. However, the calculation of the global neighbourhood and the neighbourhood bounds does not involve any floating point operations and is negligible compared to the operations performed by the slaves.

It should be noted that the master communicates information to all the slaves and expects either all slaves or only one slave to respond. The slaves only communicate with the master. This simplifies the message passing strategy that is needed.

The parallel implementation of the Kohonen algorithm described here can easily be used to implement algorithms using different adaptation rules (Eq. 3) and neighbourhoods. Examples include using a conscience mechanism [9], LVQ1 and LVQ2 [1, 8] and general competitive learning neural networks [7].

5 Training Data and Simulation Trials

The network was trained and tested by taking input values randomly from a uniformly distributed set of points from a two dimensional rectangular region. By plotting the weight vectors of the trained network, convergence was confirmed by the rectangular and uniformly distributed nature of the map's weight vectors. This is the expected behaviour of the Kohonen algorithm for this type of input.

To check the effect of communication overhead three different network sizes were tested. The size and training parameters for these three trials are shown in Table 1. Only the ordering phase of the algorithm was tested. Trial A has the smallest network size and lowest computation to communication ratio per iteration. Trial C has the largest network size and the highest computation to communication ratio per iteration.

The single chain transputer slaves of Fig. 1 were tested for runs using 1, 2, 4, 6, 8 and 10 slave transputers. The dual chain transputer slaves of Fig. 2 were tested for runs using 2, 4, 6, 8, 10 and 12 slave transputers.

Table 1: Parameter values for the three trials

Parameter	Trial A	Trial B	Trial C
n_x^g	12	60	120
n_y^g	10	50	100
iterations	50000	40000	10000
$Nh_c^{initial}$	5	25	50
$\alpha^{initial}$	0.5	0.5	0.5

6 Results

The execution times of the three different trials are shown in Table 2. The transputer configuration is indicated by kohoN_s for N_s single chain transputers ($N_t = N_s$) and koho2xN_d for dual chain transputers with N_d transputers per chain ($N_t = 2N_d$). The division in the x and y directions of the neural map is also shown. For example, (3/2) means that for the 12x10 map of Trial A 6 equal partitions of size 4x5 were formed.

Table 2: Execution times (in seconds) for the three trials

Run	Divisions	Trial A	Trial B	Trial C	Run	divisions	Trial A	Trial B	Trial C
koho1	(1/1)	192	2661	2616	-	-	-	-	-
koho2	(1/2)	143	1478	1428	koho2x1	(1/2)	173	1499	1435
koho4	(2/2)	126	876	824	koho2x2	(2/2)	158	901	829
koho6	(3/2)	126	654	593	koho2x3	(3/2)	157	676	599
koho8	(4/2)	132	530	465	koho2x4	(4/2)	164	552	467
koho10	(2/5)	140	441	371	koho2x5	(2/5)	171	464	375
-	-	-	-	-	koho2x6	(6/2)	183	428	332

From the execution times it is evident that the fastest time was for the 12 transputers in configuration koho2x6. For both Trials B and C the execution times decreased as the number of

transputers increased. However, in Trial A the execution times began to increase if more than 6 transputers were used. This can be explained by noting that Trial A uses the smallest local grid size (4x5 as noted above for 6 transputers). The communication overhead increases as more transputers are used (proportional to N_t and N_t^2) and is independent of the grid sizes. However the computations per iteration decrease as the grid sizes get smaller (inversely proportional to N_t). Thus for more than 6 transputers in Trial A the communication overhead dominates the overall execution time (which increases with more transputers). Another observation is that the dual chain transputers have a slightly longer execution for the same number of single chain transputers (e.g. compare koho6 with koho2x3). This is due to the apparent increased communication overhead involved with passing messages to a dual chain of transputers by the routing strategy adopted here (see Figs. 1 and 2).

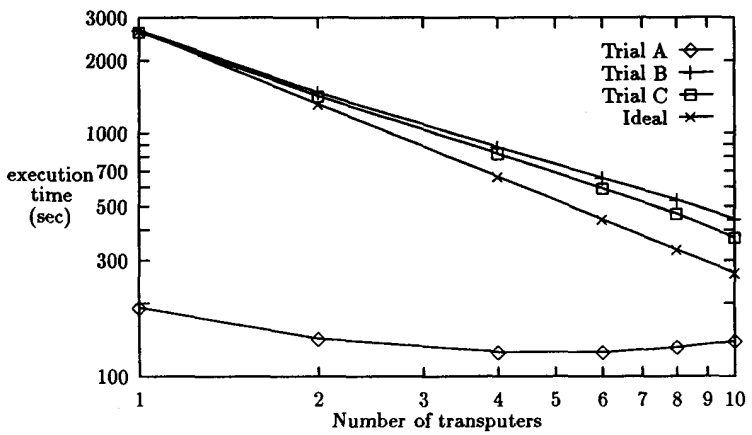

Figure 3: Log-log plot of execution times with number of transputers for a single chain configuration. The ideal plot for Trials B and C is also shown.

The effect of the communication time can be more closely examined by plotting the execution times of the single chain transputers for the three trials on the log-log graph of Fig. 3. With no communication overhead the log-log plot of execution time with number of transputers should be linear. The dominating effect of communication overhead is clearly seen from the dip in the plot for the Trial A results. Since the single transputer execution times for Trials B and C are almost identical the plots should be close. The expected ideal plot for Trials B and C is also shown and the divergence of Trials B and C with the ideal for increasing number of transputers is evident in Fig. 3. The discrepancy between the plots for Trials B and C as the number of transputers increase is due to effect of the communication overhead which is more of a problem with Trial B than C (since Trial C has the largest local grid size, the communication/computation ratio is smallest). The last T800 transputer board (used for $N_t > 8$) was of a different make and model and may result in small unexpected changes in computation or communication times as evidenced by the apparent approach of the Trial B and C results to the ideal case when 10 transputers were used.

The effect of communication overhead on the three trials will become dominant when the

computations per iteration are small. In Trial A this causes the overall execution time to actually increase. Due to the nature of the master-slave and slave-master communications necessary in the parallel implementation of the Kohonen algorithm a massively parallel implementation is not possible.

7 Conclusions

A parallel implementation of the Kohonen self-organising algorithm has been proposed and tested on a transputer farm architecture. The parallel implementation discussed here uses a farm architecture and does not require a specialised distributed operating system.

The results show that a marked improvement in execution time can be obtained by a parallel implementation. They also indicate that a massively parallel approach is not feasible since the algorithm, as it stands, requires global communication at each iteration step. It is only when the computations per transputer per iteration is sufficiently large that a parallel implementation is efficient. Thus for network of a particular size there is an upper limit to the number of parallel processes that can be run. This limit increases with increased network size. In this paper in excess of 12 transputers can be used with network sizes of the order of 3000 neurons or more but no more than 6 transputers can be used with network sizes in the order of 120 neurons. Although a more efficient routing strategy can be used this limit will still exist.

The use of master/slave or farm architecture makes the parallel implementation described here suitable for porting to any networked workstation environment where user routines to perform host to host communication exist.

The proposed parallel implementation of the Kohonen algorithm permits the achievement of very fast execution speeds as well as the ability of implementing larger sized maps.

References

[1] T. Kohonen, "Self-Organization and Associative Memory", 2nd edition, New York, Springer-Verlag 1988.

[2] T. Kohonen, "An Introduction to Neural Computing", Neural Networks, Vol. 1, pp.3-16, 1988.

[3] N.M. Nasrabadi and Y. Feng, "Vector Quantization of Images Based Uponm the Kohonen Self-Organizing Feature Maps", Proc. IEEE ICNN-88, San Diego, July 1990, pp.101-108.

[4] T. Kohonen, "The Neural Phonetic Typewriter", IEEE Computer, pp. 11-22, March 1988.

[5] Alder M., Togneri R., and Attikiouzel J. "Dimension of the Speech Space" IEE Comm, Speech and Vision. Accepted for publication.

[6] R. Mann and S. Haykin, "A Parallel Implementation of Kohonen Feature Maps on the Warp Systolic Computer", Proc. IJCNN-90, Washington D.C, Vol. 2, Jan 1990, pp 84-87.

[7] D.E. Rumelhart and J.L. McClelland (eds.), "Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations", MIT Press, Cambridge, Mass., 1986.

[8] T. Kohonen, "Improved Versions of Learning Vector Quantisation", Proc. IJCNN-90, San Diego, July 1990, pp. 545-550.

[9] D. DeSieno, "Adding a Conscience to Competitive Learning", Proc. IEEE ICNN-88, San Diego, July 1990, pp. 117-124.