[20] M. Witbrock and M. Zagha, “Backpropagation learning on the IBM GF 11”, in *Parallel Digital Implementation of Neural Networks*, pp.77-104, 1993.

[21] H. Yoon and J.H. Nang “Multilayer neural networks on distributed-memory multiprocessors”, *Proceedings of International Neural Network Conference*, Vol.2, pp.669-672, 1990.

[22] S. Zeng, “The Application of Parallel Processing Techniques to Neural Network Based Fingerprint Recognition Systems”, Thesis, Department of Electrical and Computer Engineering, West Virginia University, 1994.

[23] X. Zhang, M. Mckenna, J.F. Mesirov and D. Waltz, “An efficient implementation of the backpropogation algorithm on the Connection Machine CM-2”, in *Parallel Computing*, Vol.14, pp.317-327, 1990.

[4] E.M. Deprit, "Implementing recurrent back-propagation on the Connection Machine", in *Neural Networks*, Vol.2, pp.295-314, 1989.

[5] F. Distante, M. Sami, R. Stefanelli and G. Storti-Gajani, "Mapping neural nets onto a massively parallel architecture: a defect-tolerance solution", in *Proceedings of The IEEE*, Vol.79, pp.444-460, October 1991.

[6] S.K. Foo, P. Saratchandran and N. Sundararajan, "Analysis of training set parallelism for back-propagation neural networks", in *International Journal of Neural Systems*, Vol.6, pp.61-78, 1995.

[7] Y. Fujimoto, N. Fukuda and T. Alcabane, "Massively parallel architectures for large scale neural network simulations", in *IEEE Transactions on Neural Networks*, Vol.3, pp.876-888, November 1992.

[8] S. Haykin, *Neural Networks, a Comprehensive Foundation*, Macmillan, 1994.

[9] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.

[10] V. Kumar, S. Shekhar and M.B. Amin, "A scalable parallel formulation of the backpropagation algorithm for hypercubes and related architectures", in *IEEE Transactions on Parallel and Distributed Systems*, Vol.5, pp.1073-1089, 1994.

[11] W.M. Lin, V.K. Prasanna and K.W. Przytula, "Algorithmic mapping of neural network models onto parallel SIMD machines", in *IEEE Transactions on Computer*, Vol.40, pp.1390-1401, December 1991.

[12] Q.M. Malluhi, M.A. Bayoumi and T.R.N. Rao, "Efficient mapping of ANNs on hypercube massively parallel machines", in *IEEE Transactions on Computer*, Vol.44, pp.769-779, June 1995.

[13] A. Petrowski and G. Dreyfus, "Performance analysis of a pipelined backpropagation parallel algorithm", in *IEEE Transactions on Neural Networks*, Vol.4, pp.970-981, 1993.

[14] D.A. Pomerleau, G.L. Gusciora, D.S. Touretzky and H.T. Kung, "Neural network simulation at Warp speed: how we got 17 million connections per second", in *Proceedings of the IEEE International Conference on Neural Networks*, pp.143-150, 1988.

[15] C.R. Rosenberg and G. Blelloch, "An implementation of network learning on the Connection Machine", in *Proceedings of 10th International Conference on Artificial Intelligence*, PP.329-340, 1987.

[16] A. Singer, "Implementations of artificial neural networks on the Connection Machine", in *Parallel Computing*, Vol.14, pp.305-315, 1990.

[17] T. Yukawa and T. Ishikawa, "Optimal parallel back-propagation schemes for mesh-connected and bus-connected multiprocessors", in *Proceedings of International Neural Network Conference* pp.1748-1753, 1993.

[18] Thinking Machines Corporation, *Connection Machine CM-5 Technical Summary*, 1993.

[19] J. Tørresen, S. Mori, H. Nakashima, S. Tomita and O. Landsverk, "Exploiting multiple degrees of BP parallelism on the highly parallel computer AP1000", *Fourth International Conference on Artificial Neural Networks*, 1995.

4 Conclusions

Several parallel algorithms for a neural network based automated Fingerprint Image Comparison (FIC) system are investigated in this paper. Two types of parallelism: node parallelism and training set parallelism (TSP), as well as their combinations, are applied to the training process of the FIC system. The fingerprint training set used are obtained from a CD-ROM distributed by NIST.

Analytical results and actual measurements show that TSP has the best speedup performance among all types of parallelism. The target architecture is assumed to be a coarse-grain distributed memory parallel architecture. This type of parallelism has evenly distributed computation load, very small communication overhead. We obtained almost linear speedup up to 31.2 using 32 processors concurrently measured on a 32-node CM-5. The speedup is 2.3 times higher than the maximal speedup obtained by using node parallelism.

For the purpose of reducing the risk of slower convergence rate, a modified TSPA using weighted contributions of connections is proposed in Section 3.4.2. Our experimental results show that a fast convergence rate can be achieved by using this modified algorithm without any loss of the high speedup performance already obtained by the TSP. In this case, a significantly faster total training time can be obtained.

On the other hand, the theoretical and experimental results show that node parallelism has low speedup and bad scalability because of its significant communication overhead.

By combining the TSP with node parallelism, a better scalability can be obtained. Moreover, it can also partly reduce the risk of a slower convergence rate.

We trained the FIC system with 480 pairs of images by using the three variations of TSPA described in Section 3.4.2. The total training time and the observed accuracy for each variation are listed in Table 3. Although the observed accuracy of the weighted contribution TSP algorithm is slightly lower than the other two TSP algorithm, it has considerably reduced the total training time.

TSPAs	original	modified with equal contribution	modified with weighted contribution
total training time (hours)	49.3	15.5	5.0
recognition accuracy	96.09%	96.20%	95.49%

Table 3: Total training times and recognition accuracies of the three variations of TSPA.

References

[1] M. Annaratone, C. Pommerell, and R. Ruhl, "Interprocessor communication speed and performance in distributed-Memory parallel processors", in *Proceedings of the 16th International Symposium on Computer Architecture*, pp.315-324, 1989.

[2] P. Baldi and Y. Chauvin, "Neural Networks for Fingerprint Recognition", in *Neural Computation*, Vol.5, pp.402-418, 1993.

[3] M. Cosnard, J.C. Mignot and H. Paugam-Moisy, "Implementations of multilayer networks on parallel architectures", in *Proceedings of 2nd International Specialist Seminar Parallel Digital Processor*, 1991.

parallel implementation might be fast enough to outweight the possible slower convergence rate.

Table 2 compares the number of iterations and the total training time needed for node parallelism, original TSP and weighted contribution TSP, respectively. Three conclusions can be drawn by the examination of the table:

N_{ptn}	combined filter and image region algorithm		original TSP		weighted contribution TSP	
	iterations	total training time(hours)	iterations	total training time(hours)	iterations	total training time(hours)
32	9232	0.58	27610	0.72	245	0.0063
64	10855	1.37	119163	6.2	293	0.015
96	> 300000	> 56	266976	21	568	0.045
100	96469	18.9	102802	10.78	579	0.061
128	5915	1.5	8699	0.91	566	0.059
160	1420	0.45	3236	0.42	606	0.081
192	23462	8.9	3674	0.58	472	0.074

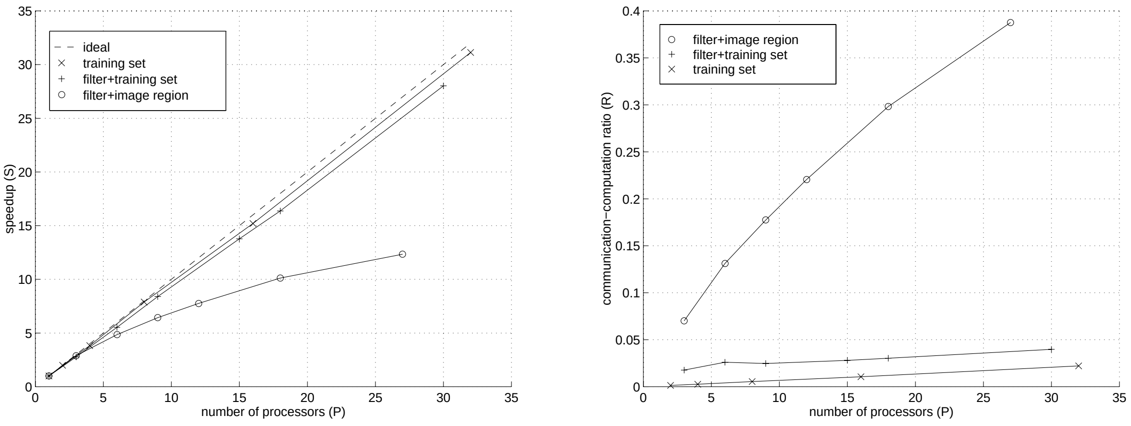
Table 2: The comparison of iterations and total training time between the node parallelism and TSP. The combined filter and image region parallel algorithm is used as the node parallelism. It runs 27 processors simultaneously. The two TSPAs run 32 processors simultaneously with $B_{ptn} = 32$. In the case of $N_{ptn} = 100$, the two TSPAs run 25 processors simultaneously with $B_{ptn} = 25$.

1. For the FIC system, the convergence rate of the node parallelism cannot be guaranteed faster than that of the TSP in all cases (i.e., $N_{ptn} = 96, 192$) although the former one has a higher weight update frequency.

2. If the node parallelism has a faster convergence rate, it can achieve a shorter total training time (i.e., $N_{ptn} = 32, 64$). However, it is much possible that the speedup of the TSP can compensate its slower convergence rate. In this case, the TSP still achieves a shorter total training time (i.e., $N_{ptn} = 100, 128, 160$).

3. The weighted contribution TSPA has both good speedup and fast convergence rate in all the cases tested. Therefore, a significantly faster total training time is obtained by this parallel algorithm.

In a small scale supercomputer, such as the 32-processor CM-5 used in our experiment, the combined parallelism cannot further improve the speedup already obtained by the TSP. However, if a well-suited node parallelism which has high efficiency (such as the filter parallelism in our application) is chosen to be combined with the TSP, the overall speedup will decrease slightly, yet a relatively higher weight update frequency can be obtained. The potential of the combined parallelism will be exposed when the neural network is implemented on a large scale supercomputer with massive processor resources. In this case, additional speedup can be gained since it can fully exploit all the available processor resources on the supercomputer.



(a)

(b)

Figure 10: Comparison of performance measurements of three parallel algorithms. (a) Speedup. (b) Communication-computation ratio.

filter and TSP. Among them, The TSP has best speedup performance with the lowest communication-computation ratio.

The node parallelism is perhaps the most intuitive approach. Nevertheless, three issues arise in terms of using this parallelism. First, the intrinsic high IPC demand of this parallel algorithm often leads to a significant communication overhead. That's the main reason why its speedup curve shown in Figure 10a is nonlinear even when the number of processors is small. Although the communication time for each pattern can be kept constant by using global reduction, it occupies a big portion of the total training time when the computation granularity becomes too fine. Figure 10b shows that the communication-computation ratio of the combined filter and image region parallelism achieves about 0.39 when 27 processors are used. This indicates that more than $\frac{1}{4}$ of the total training time is spent on communication, thus only 13.3 speedup can be achieved using this parallelism. It is predictable from Equation 17 that half of the total training time will be consumed by communication if the program is ported to a larger CM-5 machine in which all the filters and image regions can be computed in parallel. Second, it is impossible to obtain high speedup over a large number of processors. Both the first and second issues are because the computation granularity becomes too fine when many processors are used. The third problem is that in order to minimize the communication overhead and balance the computation load, some constraints will be imposed on the network decomposition method. The advantage of this algorithm is that training by pattern mode can be used for faster convergent rate.

The TSPA provides the best speedup performance among all the parallel algorithms for the FIC system. Its speedup is almost close to the ideal case. The TSP has a variety of advantages due to its coarse computation granularity. It possesses not only very even computation load and low communication demand, but also high DOP. More advantages stem from the fact that any serial steps in the training procedure can be exploited while the details of network architecture remains transparent to the programmer. Although the requirement of a large training set and a relatively slower convergence rate claim some drawback on this parallel technique, the TSP is still more attractive than the node parallelism. Because for the first issue, the real-world neural networks often demand a large amount of training data which can match the number of processors. For the second issue, each iteration of such a

- In order to increase the weight update frequency for a faster convergence rate, the block length can be shortened by combining network parallelism together, because in this case the number of local copies of the neural network on parallel computer decreases when more than one processors are used to share one copy.

An efficient hybrid parallel approach is to combine the filter parallelism and the TSP together. In this approach, every three processors corresponding to the three filters constitute a basic module. The basic module is then duplicated across all processors. So the maximal number of modules can be obtained in the 32-processor CM-5 machine is $\left[\frac{32}{3}\right] = 10$. The property of this parallel approach is similar to the pure TSP, except that after the presentation of each pattern, the processors within a module should exchange data among themselves.

The training time per epoch for this parallel implementation is:

$$T_{epoch}^{flt+tsp} = \frac{N_{ptn}}{P/P_{flt}}(T_{ptn}^{flt} + t_{com}^{flt,mod}) + \frac{N_{ptn}}{B_{ptn}}t_{com}^{tsp} \tag{26}$$

where T_{ptn}^{flt} is the training time per pattern of pure filter parallelism. It is equal to $\frac{T_{ptn}^{ser}}{P_{flt}}$. $t_{com}^{flt,mod} = 0.5971$ms is the inter-module message passing time. It is associated with the presentation of each pattern, because the filter parallelism requires data exchange for each pattern. $P_{flt} = 3$ is the number of processors assigned to the filters.

The speedup is:

$$\begin{aligned} S_{flt+tsp} &= \frac{T_{epoch}^{ser}}{T_{epoch}^{flt+tsp}} \\ &= \frac{P}{1 + R_{flt+tsp}}. \end{aligned} \tag{27}$$

The communication-computation ratio is

$$\begin{aligned} R_{flt+tsp} &= \frac{P_{flt}t_{com}^{flt,mod} + \frac{P}{B_{ptn}}t_{com}^{tsp}}{T_{ptn}^{ser}} \\ &= 1.9 \times 10^{-2} + 6.94 \times 10^{-4}P. \end{aligned}$$

By comparison with the expression of R_{tsp} of the pure TSP (Equation 22), there is an additional small constant in the expression of $R_{flt+tsp}$. However, the proportional factor remains the same order of magnitude. Thus $R_{flt+tsp}$ is still very small even when P is large. The low communication-computation ratio provides this combined algorithm a linear speedup which is close to the performance of pure training set parallelism. Meanwhile, it provides an approach which can partly overcomes the risk of slower convergence rate of the TSP. Because in this case, B_{ptn} can then be reduced to $\frac{1}{3}$ of the total available processors, as result, the weight update frequency increases three times.

3.6 Comparing the Performance of Parallel Implementations

The two types of parallelism and their combined approaches described in this section reflect distinct properties. Figure 10 shows the speedup and the communication-computation ratio of the three parallel algorithms presented in the paper: combined filter and image region parallelism, TSP, and combined

The modification of the original algorithm is mainly based on the consideration that the equal contribution to the global connection error may not be the best choice, although that is a more accurate estimate of the theory of gradient descent. In addition, in the modified algorithm, the improvement for faster convergence rate can also be observed in the experiments even without the effect of the weighted contribution. However, such improvement cannot be guaranteed in all the cases.

Table 1 shows the comparison among the convergence rates of three variations of TSPA: the original one, the modified one with *equal contribution*, and the modified one with *weighted contribution*. As mentioned above, a better convergence rate in the second algorithm can be observed from Table 1 but it cannot be guaranteed in all cases (i.e., when $N_{ptn} = 160$). Yet the third algorithm is expermentally shown to have the best convergence rate. Since the training times per iteration of three algorithm are the same, the third approach needs much less total training time than the original one.

N_{ptn}	iterations		
	original	modified with equal contribution	modified with weighted contribution
32	9685	854	245
64	119163	1690	293
96	266976	1493	568
100	102802	2555	579
128	8699	4286	566
160	3236	4395	606
192	3674	3006	472
320	21456	8170	1769
480	113625	38875	12335

Table 1: Convergence iterations for different variations of TSPA.

In principle, the coefficient c_i can be viewed as strengthening the effect of connection errors from the subsets which have larger number of unlearned patterns. After weighted by c_i, the connection errors from the subsets which have larger number of unlearned patterns dominate the convergence direction. Because generally, the convergence behavior of the subset with larger number of unlearned patterns is closer to the real convergence behavior of the whole training set. In this case, the subset with larger number of unlearned patterns is “more important” than those with smaller number of unlearned patterns. The emphasis on the “important” subsets is shown to be an efficient accelerator for faster convergence rate.

3.5 Combined Parallel Algorithms

Among the three types of parallelism presented in the previous sections, the TSP has the best speedup performance benefiting from its coarse computation granularity based on the training set decomposition. On the other hand, two cases still make the network level decomposition worthy of consideration:

- If the number of processors is large but the training set size is relatively small, which means the pure TSP may not work efficiently, further exploitation of the network parallelism can promise an additional factor of speedup.

has a speedup as high as that of the original TSPA. Thus the total training time can be largely reduced in this case.

The description of the modified algorithm is as follows:

S 1: Each P_i trains its local copy of weight matrix with the assigned training subset using *pattern mode*.

S 2: After all the patterns in the subset are presented, the local weight matrix W_i is collected from all the processors and perform the following weighted summation:

$$W = \frac{1}{P} \sum_{i=1}^{P} c_i W_i \tag{23}$$

where c_i is a coefficient attached to each W_i. It serves as as a contribution factor to the global summation.

S 3: W is broadcast back to all the processors and replaces their local weight matrices.

The local weight matrices on all processors are initialized as same, so that they are kept consistent after each epoch.

The contribution coefficient c_i is calculated from the following functions.

$$a_i = {N_i}^2 + 2N_i \tag{24}$$

$$c_i = \frac{a_i}{\sum_{i=1}^{P} a_i} \tag{25}$$

where N_i is the number of *"unlearned"* patterns in the subset whose decision outputs are beyond error tolerance of the network in processor P_i. So c_i means the contribution of the local connections to the global connection is proportional to the number of unlearned patterns in its subset. Since N_i of each subset changes during the whole training procedure, c_i also changes dynamically.

The derivation of a_i in Equation 24 is based on the experimental results. The experimental results show that if the connection errors is weighted by this function, a faster convergence rate can be achieved than some other expressions tested, such as $a_i = N_i$ or $a_i = {N_i}^2$. More accurate expressions for c_i still need to be investigated.

There are two distinctions between the original TSPA and the modified TSPA:

1. The modified algorithm tries to simulate the behavior of pattern mode weight update strategy. Each local copy of the network is trained by its training subset and updates its connections (weights) using pattern mode. At the end of epoch, each local copy submits its *connections* to the *global connections*. Instead, the original TSPA which uses gradient descent theory does not update the connections during each epoch if batch mode is being used. Each local copy submits *connection errors* to the *global connection errors*. However, the pattern mode used in the modified TSPA is only limited to its training subset. Consequently, if a block mode is used by the original TSPA and $B_{ptn} = P$, then the weight update frequency of the modified TSPA ($= \frac{N_{ptn}}{P}$) is equivalent to the weight update frequency of the original TSPA. But in this case, the message passing frequency of the original algorithm is $\frac{N_{ptn}}{P}$ times higher than that of the modified one.

2. In the original TSPA, connection errors from the local copies of the network are evenly contributed to the global connection error. While in the modified TSPA, the number of unlearned patterns in the training subset of each processor is considered as a weight factor for the connection contribution. So the modified TSPA is termed as *weighted contribution TSPA* in this paper.

$$= \frac{P}{1 + R_{tsp}} \tag{21}$$

where

$$R_{tsp} = \frac{t_{com}^{tsp}}{B_{ptn} T_{ptn}^{ser}} P \tag{22}$$

is the communication-computation ratio of TSP.

Ideally t_{com}^{tsp} remains constant when P increases[2]. It is rather small by comparison with T_{ptn}^{ser}. More prominently, the message passing requirement is once per epoch/block, which is much less frequent than that of once per pattern in node parallelism. However, this implicates lower weight update frequency.

For the case of $B_{ptn} = 32$ in our experiment,

$$R_{tsp} = 6.94 \times 10^{-4} P$$

Obviously $S_{tsp} \approx P$ even when P is very large.

The speedup of the TSP is shown in Figure 9. The maximal speedup achieves 31.2, at least 2.5 times greater than the maximal speedup obtained using node parallelism. The massive savings contributed by the concurrent processing of all the patterns provide significant scalability over typical node parallelism. All the serial steps are encapsulated and efficiently executed in parallel, thus the detailed distribution of network itself and frequent internal data exchange are thoroughly avoided.

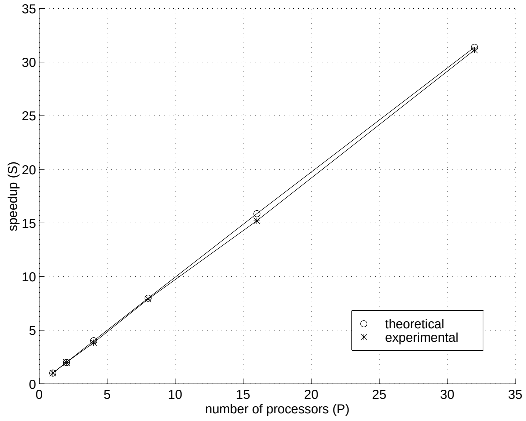


Figure 9: Theoretical and experimental speedup of training set parallelism.

3.4.2 Weighted Contribution Training Set Parallel Algorithm

The speedup of the TSP is much better than other types of parallelism presented in the paper. However, it suffers from the risk of slower convergence rate. This is due to the fact that block/batch mode which results in infrequent weight update is used in this parallelism. For the purpose of partly overcoming this drawback, the traditional TSPA is modified as described in this section. The experiments prove that the modified algorithm is valuable: it significantly reduces the iterations for convergence, meanwhile, it

[2] Due to some random features in time-sharing operation system and processors architecture, the computation times among the processors are not exact the same even if their loads are identical. This computation delay among the processors will lead to additional communication overhead especially when the number of processors is large.

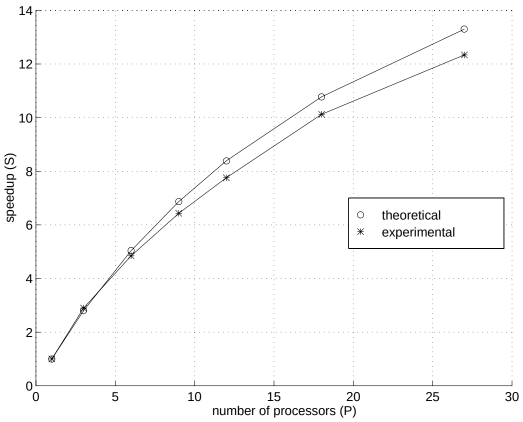


Figure 7: Theoretical and experimental speedup of the algorithm combining the filter and image region parallelisms.

update strategy. The block length (number of patterns in each block) should be larger than the number of available processors to be able to fully utilize all the processor resources.

3.4.1 Implementation of Training Set Parallel Algorithm

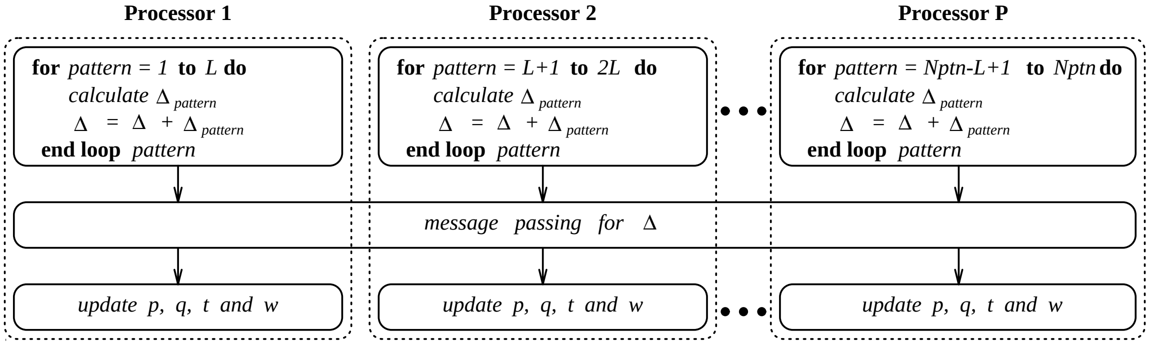


Figure 8: Procedure diagram for training set parallelism. $\Delta = \{\Delta W, \Delta t, \Delta p, \Delta q\}$. The number of patterns in each subset is determined by $L = N_{ptn}/P$.

The diagram of the training set parallel algorithm (TSPA) is shown in Figure 8. It runs many copies of the FIC system concurrently by the distribution of the training set across the processors. The training time for an epoch is given by

$$T^{tsp}_{epoch} = \frac{N_{ptn}}{P} T^{ser}_{ptn} + \frac{N_{ptn}}{B_{ptn}} t^{tsp}_{com} \tag{20}$$

where t^{tsp}_{com} is the communication time for *one epoch.* It is measured as 2.094ms per epoch.

The speedup of the TSP is

$$S_{tsp} = \frac{T^{ser}_{epoch}}{T^{tsp}_{epoch}}$$

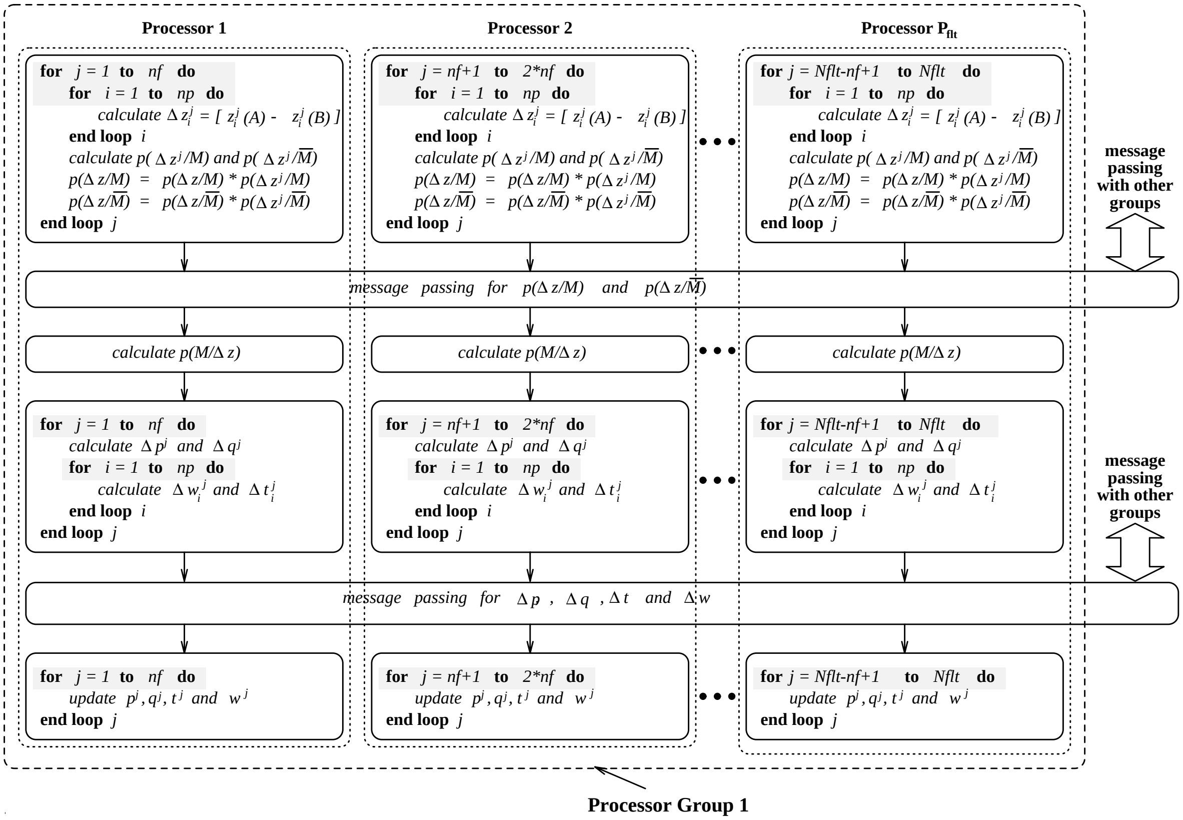

Processor 1
Processor 2
Processor P_{flt}
for $j = 1$ to nf do
for $i = 1$ to np do
calculate $\Delta z_i^j = [\, z_i^j(A) - z_i^j(B)\,]$
end loop i
calculate $p(\Delta z^j/M)$ and $p(\Delta z^j/\overline{M})$
$p(\Delta z/M) = p(\Delta z/M) * p(\Delta z^j/M)$
$p(\Delta z/\overline{M}) = p(\Delta z/\overline{M}) * p(\Delta z^j/\overline{M})$
end loop j
for $j = nf+1$ to $2*nf$ do
for $j = Nflt-nf+1$ to $Nflt$ do
message passing with other groups
message passing for $p(\Delta z/M)$ and $p(\Delta z/\overline{M})$
calculate $p(M/\Delta z)$
calculate Δp^j and Δq^j
calculate Δw_i^j and Δt_i^j
message passing for Δp, Δq, Δt and Δw
update p^j, q^j, t^j and w^j
Processor Group 1

Figure 6: Procedure diagram for combined filter and image region parallelism. Both the filters and the image regions are processed in parallel. Only Processor Group 1 is shown in the figure. P_{flt} is the number of processors per group, $nf = \dfrac{N_{flt}}{P_{flt}}$, and $np = \dfrac{N_{ptch}}{P/P_{flt}}$. The shadowed areas show how the computation load is distributed to processors.

Its *communication-computation ratio* $R_{flt+reg}$ is

$$R_{flt+reg} = \frac{P}{0.95 + 3.97 \times 10^{-3} P}. \tag{19}$$

In Equation 17 to 19, P is the total number of processors being used. P_{flt} is the number of processors assigned to the computation of the filters for a given image region. Obviously it is the most efficient to run all the filters in parallel, thus P_{flt} can be fixed as equivalent to N_{flt}. The communication time $t_{com}^{flt+reg}$ is 2.2104ms per pattern. The corresponding theoretical and experimental results are shown in Figure 7. The speedup achieves 13.3 by using 27 processors simultaneously.

3.4 Training Set Parallel Algorithms for the FIC System

Training set parallelism (TSP) is a coarse computation granularity approach. It exploits parallelism at higher level of abstraction. By contrast, the node parallelism which requires network decomposition is termed as low level partitioning scheme. In TSP, the neural network is duplicated on each processor. A subset of the training set is allocated to each processor. Training by batch/block is required as weight

To avoid the slow I/O operation during the training cycle, all the training patterns are stored in memory since there is sufficient local memory in each processor. Thus downloading time is only needed initially. That means the downloading time is very short compared to the total training time and is eliminated in the time analyses.

3.3 Node Parallel Algorithms for the FIC System

There are several ways to partition the FIC system into a set of slices which can be executed in parallel. In order to balance the computation load among different processors, the network should be decomposed as even as possible. Three different decompositions can be considered to solve this problem.

Parallelism among different filters. In the FIC system, the computation involved with each filter are exactly the same. Thus one approach of decomposition is to distribute the filters across the processors. A limitation of this approach is that the maximum speedup is restricted by the number of the filters used in the FIC system. Although more filters can be introduced to improve the reliability in the recall phase and increase the speedup, there will not be a large number of filters assumed in the FIC system.

Parallelism among different image regions. Parallelism on smaller computation granularity can be exploited to obtain a higher degree of parallelism (DOP, which is defined as the number of processors used to execute a parallel program). A simple method is to decompose the image into several regions with the same size, and distribute them across the processors. The computation load involved with this image can then be evenly distributed across those processors. An image consists of 36 patches, each of which has independent computation with the filter set. An image region should consist of multiple number of patches in order to avoid frequent internal data exchange. Since the possible number of image regions is much larger than that of the filters, more processors can be used concurrently to achieve a higher speedup in comparison with the filter parallelism.

Combined parallelism among different filters and image regions. To further improve the speedup through the node level parallelism, the computation granularity can be made even smaller by combining above two decomposition approaches together. In this case, each processor is responsible for the computation involved in a group of image patches and a specified (or a subset of) filter(s).

The detailed implementation of combined parallelism among filters and image regions is discussed below. The partitioning among different filters and image regions can be combined together to use more processors on the target machine. The processors are split into several groups. Each group is responsible for a set of image patches and within the group different processors correspond to different filters. The communication demand is the same as that in image region parallelism. An algorithm diagram for one such group is shown in Figure 6.

The training time for an epoch and the speedup in this implementation are given by

$$T_{epoch}^{flt+reg} = N_{ptn}[\frac{N_{ptch}N_{flt}}{P}t_{fp} + \frac{N_{flt}}{P_{flt}}t_f + t_{com}^{flt+reg}] \qquad (17)$$

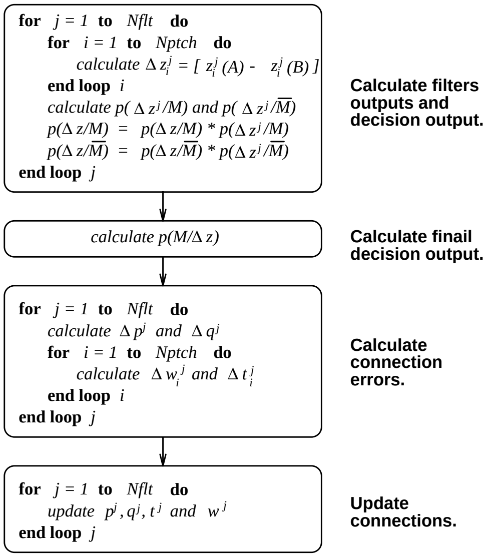
$$\begin{aligned} S_{flt+reg} &= \frac{T_{epoch}^{ser}}{T_{epoch}^{flt+reg}} \\ &= \frac{P}{0.952 + 3.97 \times 10^{-2}P}. \qquad (18) \end{aligned}$$

Both node parallelism and pipelining approach try to partition the neural network into several pieces, and assign a piece of network to one or more processors working with whole training set. While the training set parallelism tries to partition the training set across the processors but encapsulate an entire copy of neural network.

3.2 Time Analysis of the Sequential Training Algorithm for the FIC system

For the sake of the speedup analysis in the parallel implementations, the response time of the FIC system sequential training algorithm is discussed first.

Figure 5 shows the sequential training procedure when the preprocessed data of one pattern (one pattern refers to a pair of reference and test images, see Section 2) is submitted. The pattern mode is used as weight update strategy.



```
for $j = 1$ to $Nflt$ do
    for $i = 1$ to $Nptch$ do
        calculate $\Delta z_i^j = [ z_i^j (A) - z_i^j (B) ]$
    end loop $i$
    calculate $p(\Delta z^j/M)$ and $p(\Delta z^j/\overline{M})$
    $p(\Delta z/M) = p(\Delta z/M) * p(\Delta z^j/M)$
    $p(\Delta z/\overline{M}) = p(\Delta z/\overline{M}) * p(\Delta z^j/\overline{M})$
end loop $j$

calculate $p(M/\Delta z)$

for $j = 1$ to $Nflt$ do
    calculate $\Delta p^j$ and $\Delta q^j$
    for $i = 1$ to $Nptch$ do
        calculate $\Delta w_i^j$ and $\Delta t_i^j$
    end loop $i$
end loop $j$

for $j = 1$ to $Nflt$ do
    update $p^j, q^j, t^j$ and $w^j$
end loop $j$
```

Calculate filters outputs and decision output.
Calculate finail decision output.
Calculate connection errors.
Update connections.

Figure 5: Sequential procedure diagram for training the FIC system.

The training time for one pattern T_{ptn}^{ser} consists of several time components corresponding to the various computation steps in forward and backward passes of the training procedure given by Equations 2- 14. More than 99% of the components are directly proportional to the number of filters (denoted by N_{flt}) and/or the number of image patches (denoted by N_{ptch}). Thus T_{ptn}^{ser} can be written as

$$T_{ptn}^{ser} \approx N_{flt} N_{ptch} t_{fp} + N_{flt} t_f \qquad (15)$$

where $t_{fp} = 0.8305$ms is the time constant related to both filters and image patches, and $t_f = 1.5286$ms is the time constant related to filters only. They are measured by running the program on a single processor of the CM-5.

The sequential training time for *one epoch* is thus given by

$$T_{epoch}^{ser} = N_{ptn} T_{ptn}^{ser} \qquad (16)$$

where N_{ptn} is the number of patterns in the training set.

3. It has a faster convergence rate since it updates the weights N_{ptn} times more frequently then the batch mode, where N_{ptn} is the number of patterns in the training set.

On the other hand, the batch mode is well suited to parallelization because weight errors can be calculated independently. To improve the convergence rate of batch mode, the weights may have to be updated more frequently than once per epoch. Thus a hybrid update strategy, the block mode, is adopted in many parallel implementations of neural network.

The parallelism inherent in the multilayer ANN paradigm itself reveals several different types of parallelism [16, 19]:

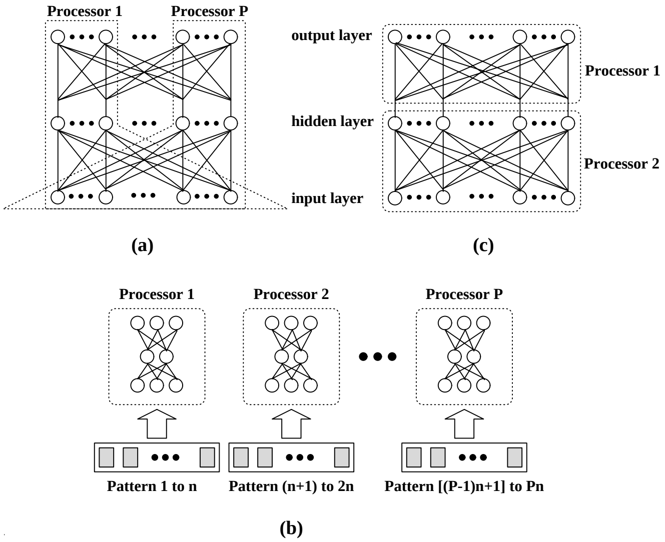


Figure 4: Three types of parallelism inherent in ANN paradigm: (a) Node parallelism. (b) Training set parallelism. (c) Pipelining. The dotted lines show how the network is mapped to processors.

.

- **Node Parallelism** The nodes at the same layer are partitioned and distributed over the processors. Each processor concurrently simulates a slice of the network over the whole training set (Figure 4a). Further, the computation within each node may also run in parallel. In this method, the weights can be updated by using *pattern mode.*

- **Training Set Parallelism** The training set is partitioned into several subsets, while the whole neural network is duplicated on each processor. Each processor works with a subset of the training set (Figure 4b). The weights are updated by using the *block mode* or the *batch mode.*

- **Pipelining** The forward and backward passes of different training patterns can be pipelined, i.e., while the output layer processor calculates output and error values, the hidden layer processor concurrently processes the next training pattern (Figure 4c). Pipelining requires a delayed weight update[1] or *block/batch mode.*

[1] The delayed weight update mode is a variation of pattern mode which is especially designed for pipelining implementation. In this mode, weight update is delayed after the presentation of several patterns.

3 Parallel Algorithms for the FIC System

The various parallel approaches investigated in the paper are targeted for a coarse grain distributed memory parallel architecture. A 32-node CM-5 is used as the platform for implementations and measurements.

In order to develop efficient parallel algorithms for the FIC system, a careful examination of the parallelism inherent in the ANNs themselves is fundamental. Three types of parallelism: node parallelism, training set parallelism (TSP) and pipelining, as well as their combinations are investigated in this section. The results show that significant speedup can be obtained through the efficient exploration of parallelism inherent in the FIC system.

3.1 Parallel Paradigms for ANNs

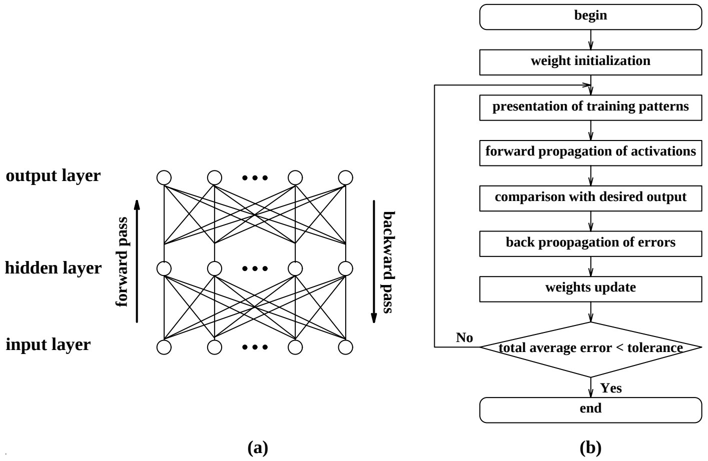

output layer
hidden layer
input layer
forward pass
backward pass
begin
weight initialization
presentation of training patterns
forward propagation of activations
comparison with desired output
back proopagation of errors
weights update
No
total average error < tolerance
Yes
end
(a)
(b)

Figure 3: (a) A three-layer ANN. (b) Backpropagation training procedure.

A typical backpropagation algorithm for a multilayer ANN is described by Figure 3. An important feature which affects the design of a parallel neural network algorithm is the *weight updating strategy*. Basically there are three different strategies described as follows:

- **Pattern Mode Strategy** Weight updating is performed after the presentation of each training pattern.

- **Batch Mode Strategy** Weight updating is performed after the presentation of all the training patterns that constitute an epoch.

- **Block Mode Strategy** Weight updating is performed after the presentation of a subset of the training patterns.

The pattern mode has some advantages over the batch mode [8]:

1. It requires less local storage for the weights.

2. The patterns can be presented to the network in a stochastic manner, which makes some training algorithms, i.e., the backpropagation, less likely to be trapped in a local minimum.

Summarily, the adjustable parameters of the neural network are $w^j_{x,y}$, t^j, p_j, and q_j. In this implementation, their total number is $(7 \times 7 + 1) \times 3 + 2 \times 3 = 156$. The parameters are initialized randomly within a small range. They are then iteratively adjusted after the presentation of each image pair, by gradient descent on the cross entropy error measure H. The equations for calculating parameter errors are given below.

Assume $p = p(M/\Delta z)$. Let

$$c = \begin{cases} -(u/v) \times p & \text{if } P = 1 \\ -(u/v) \times p^2/(p-1) & \text{if } P = 0 \end{cases} \tag{9}$$

$$c_w = 2c \ln \frac{(1-p_j)(1-q_j)}{p_j q_j} \tag{10}$$

where

$$u = p(\Delta z/\overline{M})p(\overline{M})$$
$$v = p(\Delta z/M)p(M).$$

Then the cross entropy errors of adjustable parameters can be derived from Equations 3- 8:

$$\frac{\partial H}{\partial p_j} = c(n/p_j - \frac{1}{p_j(1-p_j)} \sum_i \Delta z^j_i) \tag{11}$$

$$\frac{\partial H}{\partial q_j} = c(n/q_j - \frac{1}{q_j(1-q_j)} \sum_i \Delta z^j_i) \tag{12}$$

$$\begin{aligned} \frac{\partial H}{\partial w^j_{x,y}} &= c_w \sum_i \{[z^j_i(A) - z^j_i(B)][z^j_i(A)(1 - z^j_i(A))I_{rs}(A) - \\ &\quad z^j_i(B)(1 - z^j_i(B))I_{rs}(B)]\} \end{aligned} \tag{13}$$

$$\frac{\partial H}{\partial t^j} = c_w \sum_i \{[z^j_i(A) - z^j_i(B)][z^j_i(A)(1 - z^j_i(A)) - z^j_i(B)(1 - z^j_i(B))]\}. \tag{14}$$

The summation of i is taken from 1 to 36, the total number of patches per image. $j = 1, 2, 3$ is the number of filters used in the FIC system.

By gradient decent method,

$$\begin{aligned} p_j(k+1) &= p_j(k) - \eta \frac{\partial H}{\partial p_j}; \\ q_j(k+1) &= q_j(k) - \eta \frac{\partial H}{\partial q_j}; \\ w^j_{x,y}(k+1) &= w^j_{x,y}(k) - \eta \frac{\partial H}{\partial w^j_{x,y}}; \\ t^j(k+1) &= t^j(k) - \eta \frac{\partial H}{\partial t^j} \end{aligned}$$

where $\eta \ll 1$ is the learning rate.

After the neural network part of the FIC system is trained by 100 image pairs, a recognition accuracy of 93.78% is obtained. One approach to further improve the recognition accuracy is to increase the training set size. However, it will be very time consuming if the network is trained on serial computer with a large training set. In this case, the parallel implementation becomes mandatory.

training. Denote i as the position of (x, y). For the outputs of each filter j, the squared difference between two images is

$$\Delta z_i^j(A, B) = [z_i^j(A) - z_i^j(B)]^2. \tag{2}$$

Denote $\Delta z = \Delta z(A, B)$ as the array of all $\Delta z_i^j(A, B)$ for all position i and filter j.

2.2.2 Decision

The purpose of the decision part of the network is to estimate the probability $p = p[M(A, B)/\Delta z(A, B)] = p(M/\Delta z)$ of a match between images A and B, given the evidence Δz provided by the convolution filters. The decision part can be viewed as a binary Bayesian classifier. There are four key ingredients to the decision network:

1. Because the output of the network is to be interpreted as a probability, the usual least mean square error used in most backpropagation networks does not seem to be an appropriate measure of network performance. For probability distributions, the cross entropy between the estimated probability output p and the true probability P, summed over all patterns, is a well-known information theoretic measure of discrepancy

   $$H(P, p) = \sum_{(A,B)} (P \log \frac{P}{p} + Q \log \frac{Q}{q}) \tag{3}$$

   where, for each image pair, $Q = 1 - P$ and $q = 1 - p$.

2. Using Bayes inversion formula and omitting, for simplicity, the dependence on the image pair (A, B)

   $$p(M/\Delta z) = \frac{p(\Delta z/M)p(M)}{p(\Delta z/M)p(M) + p(\Delta z/\overline{M})p(\overline{M})} \tag{4}$$

   and choosing appropriate values for $p(M)$ and $p(\overline{M})$.

3. Making the simplifying independence assumption that

   $$p(\Delta z/M) = \prod_{i,j} p(\Delta z_i^j/M) \tag{5}$$

   $$p(\Delta z/\overline{M}) = \prod_{i,j} p(\Delta z_i^j/\overline{M}). \tag{6}$$

   In the center of a fingerprint where there is more variability than in the periphery, it is a reasonable approximation, which leads to a simple network architecture and, with hindsight, yields excellent results.

4. Using binomial model for $p(\Delta z_i^j/M)$ and $p(\Delta z_i^j/\overline{M})$:

   $$p(\Delta z_i^j/M) = p_j^{1-\Delta z_i^j}(1 - p_j)^{\Delta z_i^j} = p_j(\frac{1 - p_j}{p_j})^{\Delta z_i^j} \tag{7}$$

   $$p(\Delta z_i^j/\overline{M}) = q_j^{\Delta z_i^j}(1 - q_j)^{1-\Delta z_i^j} = (1 - q_j)(\frac{q_j}{1 - q_j})^{\Delta z_i^j} \tag{8}$$

   with $0.5 \leq p_j, q_j \leq 1$. This is again only an approximation used for its simplicity and for the fact that the feature differences Δz_i^j are usually close to 0 or 1. In this implementation and for economy of parameters, the adjustable parameters p_j and q_j depend only on the filter. In a more general case they could also depend on location.

2.2 Neural Network Decision Stage

The decision stage is the neural network part of the algorithm. The network has a pyramidal architecture, with two aligned and compressed central regions as inputs along with a single output p (see Figure 2). The bottom level of the pyramid corresponds to a convolution of the compressed central regions with respect to a set of filters or feature detectors. The decision layer implements results from a probabilistic Bayesian model for the estimation of p, based on the output of the convolution filters. Both the filtering and decision part of the network are adaptable and trained simultaneously.

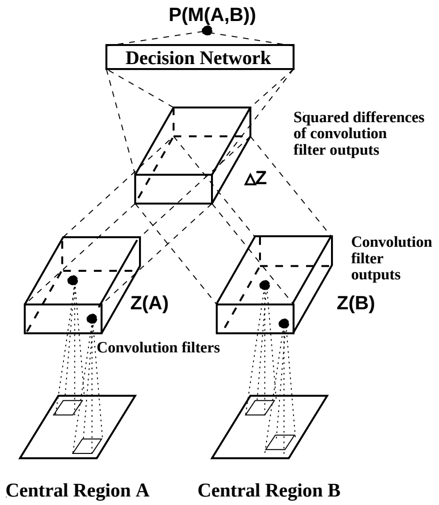


Figure 2: Neural network architecture [2].

2.2.1 Convolution

The two central regions are first convolved with a set of adjustable filters. Three different filters are used in our application. Each filter has a 7×7 receptive field and moves across all the image by 5 pixels each step.The two neighboring receptive fields have an overlap of 2 pixels to approximate a continuous convolution operation. Thus each 32×32 compressed central region is transformed into three 6×6 output arrays, one for each filter. The output of filter j at position (x, y) in one of these arrays is given by (for instance for A):

$$z^j_{x,y}(A) = f(\sum w^j_{x-r,y-s} I_{r,s}(A) + t^j_{r,s}) \tag{1}$$

where $I_{r,s}(A)$ is the pixel value in the compressed central region of image A at the position of (r, s), f is the sigmod function

$$f(x) = \frac{1}{1 + e^{-x}},$$

$w_{x-r,y-s}$ is the weight of synaptic connection from the position of (r, s) in the compressed central region to the position of (x, y) in the array of filter outputs, and t^j is a threshold. The summation in Equation 1 is taken over the 7×7 patch coincident with the receptive field of the filter at the (x, y) location. The threshold and the weights of 7×7 receptive field are characteristic of the filter, so that they can also be viewed as the parameters of a translation invariant convolution kernel. They are shared within an image but also across the images A and B. These $7 \times 7 + 1 = 50$ parameters are adjustable during network obtained by these implementations proves that highly general purpose parallel computers can compete with the performance of special purpose computers while avoiding dedicated hardware elements with dedicated connection paths.

In 1994, we developed a neural network based automated Fingerprint Image Comparison (FIC) system [22]. The purpose of the development is to meet the rapidly growing load of fingerprint recognition in FBI Identification Division. When presented with a pair of fingerprint images, the FIC system outputs an estimate of the probability that the two images originate from the same finger. The preliminary test showed 93.78% recognition accuracy after the system was trained by 100 pairs of images.

The FIC system was first implemented on SUN Sparc workstation. The training time for 100 pairs of images was about 6 to 8 days. Such a long time made it difficult to extend the training set for higher recognition accuracy.

The goal of this paper is to develop efficient parallel algorithms by exploiting parallelism inherent in the FIC system. In order to do this, node parallelism, training set parallelism (TSP), and combined parallelism are examined. Theoretical analyses of the performance of various parallel algorithms are performed. To obtain better speedup performance and to overcome the drawback of TSP, several new extensions are investigated.

The paper is organized into four sections. Section 2 describes a neural network based FIC system architecture. Section 3 presents a variety of parallel algorithms for the FIC system. The performances of these algorithms are compared based on the theoretical analyses and the experimental results of the implementations. Section 4 summarizes the main conclusions of this paper.

2 The FIC System Architecture

The neural network based Fingerprint Image Comparison (FIC) system presented in this section is mainly based on an algorithm described by Baldi and Chauvin in [2] and modified in [22]. The system consists of two components: a preprocessor and a neural network based decision stage (Figure 1). When presented with a pair of fingerprint images, the system outputs an estimate of the probability that the two images originate from the same finger. The fingerprints are obtained from the NIST (National Institute of Standards and Technology) Fingerprint Database CD-ROM which contains 4000 (2000 pairs, two different rollings of the same finger) 8-bit grey scale images of randomly selected fingerprints. Each print is 512 × 512 pixels with 32 rows of white space at the bottom of the print.

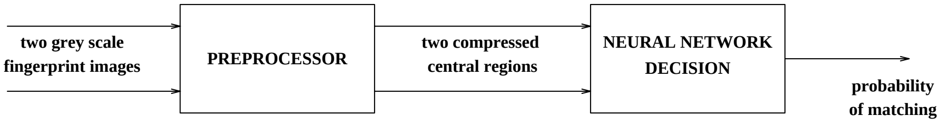


Figure 1: Operations of the FIC system.

2.1 Preprocessing Stage

The purpose of the preprocessing stage is to extract a central region from each one of the two input images, and to align the two central regions. The outputs of the preprocessor are two compressed central regions with 32 × 32 pixels each. The detailed algorithm of preprocessor is described in [22].

Parallel Algorithms for an Automated Fingerprint Image Comparison System

Hany H. Ammar and Zhouhui Miao
Department of Electrical and Computer Engineering
West Virginia University, Morgantown, WV 26506, USA

October 6, 1995

Abstract

This paper addresses the problem of developing efficient parallel algorithms for the training procedure of a neural network based Fingerprint Image Comparison (FIC) system. The target architecture is assumed to be a coarse-grain distributed memory parallel architecture. Two types of parallelism: node parallelism and training set parallelism (TSP) are investigated. Theoretical analysis and experimental results show that node parallelism has low speedup and poor scalability, while TSP proves to have the best speedup performance. TSP, however, is amenable to a slow convergence rate. In order to reduce this effect, a modified training set parallel algorithm using weighted contributions of synaptic connections is proposed. Experimental results show that this algorithm provides a fast convergence rate, while keeping the best speedup performance obtained.

The combination of TSP with node parallelism is also investigated. A good performance is achieved by this combiation approach. This provides better scalability with the tradeoff of a slight decrease in speedup.

The above algorithms are implemented on a 32-node CM-5.

1 Introduction

A large scale artificial neural network (ANN) used to solve the real-world application problems, such as pattern recognition, vision and speech recognition, can require enormous amounts of computational resources. The mass of processing elements and synaptic connections that comprise an ANN, as well as a generally required large training set, make ANN inherently computation intensive. In fact, most of the serial computers available today are bound to be overwhelmed by the computational demands of ANNs for real-world application [16].

Generally, the target machines for implementing parallel neural network simulations fall into two categories: special-purpose parallel processors and general-purpose parallel computers. For the special-purpose parallel processors, they have the advantages of very high processing speed, low cost and small size, whereas it loses the flexibility once a neural network has been "cast" on it.

On the other hand, general-purpose parallel computers can be applied to a wide variety of neural network models. In these applications, the parallel models are created in software while some compromise is provided between the flexibility and the speedup.

Some widely used training algorithms for multilayer neural networks, especially the backpropagation algorithm, have been implemented on several types of supercomputers, such as the Warp [14], the Connection Machine [15, 16, 23], the IBM GF 11 [20], and the Fujitsu AP1000 [19]. The speedup

Contents