Strategies for Parallelizing Supervised and Unsupervised Learning in Artificial Neural Networks Using the BSP Cost Model

R.O. Rogers and D.B. Skillicorn
{rogers,skill}@qucis.queensu.ca

June 1997
External Technical Report
ISSN-0836-0227-
97-406

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Document prepared June 2, 1997

Abstract

We use the cost system of BSP (Bulk Synchronous Parallelism) to predict the performance of three different parallelization techniques for both supervised and unsupervised learning in artificial neural networks. We show that exemplar parallelism outperforms techniques that partition the neural network across processors, especially when the number of exemplars is large, typical of applications such as data mining.

Keywords: Bulk synchronous parallelism, BSP, batch learning, supervised learning, unsupervised learning, backpropagation, competitive layer network, cost metrics, performance evaluation, Cray T3D, Cray T3E, IBM SP2, SGI PowerChallenge.

1 Introduction.

Training neural networks is time-consuming, so the possibility of applying parallelism to it has been extensively considered. Three approaches have been used. The first is to build special-purpose hardware implementations of neural networks [10, 11, 14]. The second is to map neural networks onto parallel computers with particular, regular communication topologies [4, 13, 15] in ways that minimize the communication required. This approach has become less important because, increasingly, parallel computers use less-well-structured communication topologies. The third approach is to develop neural network implementation strategies for general parallel computers [3]. A good analysis of general-purpose approaches as part of the Promoter project can be found in [2].

It is difficult, in practice, to determine the best way to exploit parallelism because of the number of different ways in which to divide the work of training into concurrently-executable pieces, the communication-intensive nature of training algorithms, and the absence of realistic analytic ways of predicting the performance of parallel computers.

A new parallel computation model, Bulk Synchronous Parallelism (BSP), is designed for communication-intensive applications, and has a cost system that is both simple and accurate on a wide range of real machines [12]. The BSP model is a perfect fit with the computational structure of neural network training. The predictions of its cost model are typically within a few percent of the actual performance achieved [7], so comparisons of implementation strategies can be made without laborious implementation and testing.

We express some obvious strategies for neural network parallelization in a BSP style, and then use the cost model to predict their performance. We conclude that, for both supervised and unsupervised learning, exemplar parallelism is a clear win. This agrees with intuition and with the results of previous implementations. Our contribution is to show that these performance relationships are not artifacts of particular implementation techniques, but are intrinsic, arising from the dense communication requirements of neural network training. We also quantify how poorly certain parallelization techniques must perform, and thus provide evidence to remove them from practical consideration.

In the next section, we provide a brief introduction to BSP. In Section 3, we describe parallelization strategies for neural network learning. In Section 4, we analyse supervised learning (multilayer perceptrons), and in Section 5 we analyse unsupervised learning (self-organizing feature maps).

2 Bulk Synchronous Parallelism.

A BSP abstract machine consists of a set of processor-memory pairs connected by a communication topology. It is straightforward to emulate such an abstract machine on any MIMD architecture [12].

BSP programs are sequential compositions of *supersteps*. A superstep is a parallel construct in which each processor executes the same code. There are three phases to each superstep. They are:

1. Local computation in each processor, using only local variables;

Computer	Processors	Exec Rate (Mflops)	g (flops/word)	l (flops)
SGI PowerChallenge	4	74	0.5	1902
Cray T3D	4	12	0.8	168
	16	12	1.0	181
	64	12	1.7	148
	256	12	2.4	387
Cray T3E	4	46.7	1.8	357
	16	46.7	1.7	751
IBM SP2	4	26	8.0	3583
	8	26	11.4	5412
Sun	4	10.1	4.1	118

Figure 1: BSP Parameters for Typical Parallel Computers.

2. Global communication among processors;

3. A barrier synchronisation, after which all communicated values become visible in local memories.

The key to BSP's simple style, accurate cost prediction, and high performance are:

- Separating communication from synchronisation, which makes it impossible to write programs that deadlock.

- Treating communication as global, rather than the aggregate of many point-to-point communications. This makes congestion effects visible, so that they can be minimized by the runtime system, and their cost reflected for software developers.

The cost of a superstep can be straightforwardly computed from the program text and two architectural parameters. The first of these, g, measures the ability of an architecture's communication network to deliver continuous data. It is in units of floating point operation times to deliver each 32-bit word, so a global communication phase in which each processor sends and receives no more than h words completes in the same time as hg floating point instructions. This use of a single parameter to model an apparently-complex phenomenon works because today's parallel computers take very little advantage of locality; the cost of communication is dominated by the time to cross the boundary between processors and network. The second parameter, l, measures the time taken for barrier synchronisation. Figure 1 shows typical values of these parameters for some popular parallel computers.

Consider a superstep in which processor i takes w_i instructions for its local computation and sends and receives h_i words. The cost of the superstep is the sum of the costs of its three phases, and so is:

$$\mathop{\mathsf{MAX}}_{\text{processes}} w_i \; + \; \mathop{\mathsf{MAX}}_{\text{processes}} h_i \, g \; + \; l \qquad (1)$$

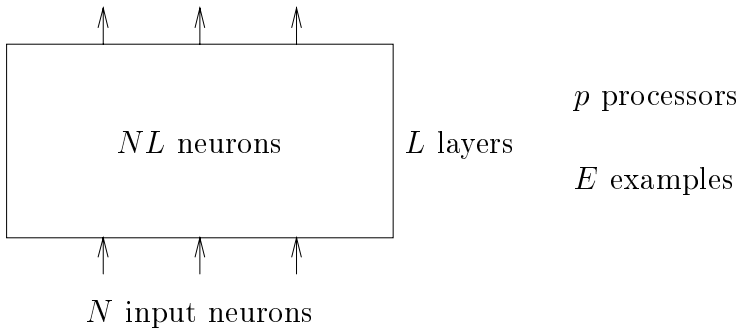

Figure 2: Neural Network Parameters.

The units of this expression are floating point operation times, which can be converted to time by dividing through by the instruction execution rate. This cost formula is not just a theoretical approximation – it has been shown to be highly accurate across a wide range of architectures and applications [7].

3 Parallelization Strategies for Learning.

For simplicity and generality, we consider neural networks with L layers, and N neurons per layer, with each layer in the network fully-connected to the preceding and succeeding layers. The total number of neurons is therefore NL. We assume this rectangular structure because all other network connectivities are subsets of this topology, and it represents the worst-case communication density. We also assume that p processors are used, and that the training set consists of E examples. This is illustrated in Figure 2.

Three parallelization strategies are considered. They are:

1. *Exemplar parallelism (EP).* This approach uses the existence of large numbers of exemplars as the source of parallelism. It does not attempt to exploit the parallelism present in a neural network itself.

   Each processor is given a subset of the exemplars, and trains on each, using the standard sequential technique. The weight corrections are accumulated in each processor, as in sequential batch learning. When each processor has processed its exemplars, the weight corrections are summed across processors to give a single, global correction to the weight space.

2. *Block parallelism (BP).* This approach uses the parallelism within the neural network, partitioning the network into blocks of connected neurons. We assume that these are non-overlapping rectangular blocks of dimension x deep by y wide. This exploits the (rather limited) locality present in the connections between neurons.

   Each block is assigned to a processor. Exemplars are presented to those processors holding blocks containing the input layer of the neural network. These processors execute the standard backpropagation algorithm on the neurons they contain, passing

activations on to the processors holding layers $x+1$, $x+2$, and so on. Errors propagate backwards in the obvious, reverse way. Note that rows of processors behave as a two-way pipeline.

3. *Neuron parallelism (NP).* This approach also uses the parallelism within the neural network, but does not attempt to exploit locality. Instead, neurons are randomly allocated to processors.

   Each processor is assigned n/p neurons. Standard backpropagation is used, as in block parallelism, except that the random allocation means that there is no geometric relationship between processors. Instead, updates are done on a data-driven basis. Inter-processor communication occurs whenever an activation or error must move between neurons allocated to different processors.

All three of these parallel implementation techniques fit well with BSP because load balancing can be statically optimised. In all three cases, there is an obvious way to ensure that each processor is allocated the same amount of work. Although block parallelism exploits locality, every processor communicates equally with every other processor holding part of the same layer of the neural network; thus locality-based models will not outperform BSP significantly. As a pragmatic matter, BSP performs as well, or better, than models such as MPI or PVM [8, 9].

4 Cost Analysis for Supervised Learning.

In this section, we examine the relative costs of the three parallelization techniques for supervised learning using multilayer perceptrons. Observe that the number of weights that must be updated to learn one exemplar is $W = N^2(L-1)$. We will assume that it takes A operations to update each weight.

For exemplar parallelism, learning can be divided into two phases: the learning of a subset of exemplars by each processor independently, and then the global computation of the weight correction. Thus EP is implemented using two supersteps.

The first superstep requires each processor to compute W/p weight updates for each exemplar. Thus the computation cost is AEW/p. At the end of this superstep, each processor passes its weight space correction (of size W) to a single processor. The cost of this communication is $(p-1)Wg$.

Naively it might seem that merging weight space corrections using a tree would be preferable, since this takes (in theory) logarithmic time. Indeed, some implementations do this [3, 14]. However, experimental results [6] show that reductive trees perform very poorly on today's architectures because of the poor performance of synchronisations. A BSP barrier synchronisation is likely to be *faster* than even a single lock or semaphore on real machines. As a result, the logarithmic number of steps of the reductive algorithm does not produce the best overall performance.

The computation performed by this single processor to compute the final weight space correction from the weight corrections received from each of the other processors takes time

$(p-1)W$. Thus, the overall BSP cost of this parallel implementation, using Equation 1, is:

$$\left[\frac{AEW}{p} + (p-1)W\right] + [(p-1)W]\, g + 2l$$

For block parallelism, the superstep structure is somewhat complicated. Recall that each processor is responsible for a block of neurons of depth x and width y. It is easy to see that $xyp = NL$. The processors can be regarded as arranged in rows, the first handling neurons in layers 1 to x, the second handling neurons in layers $x+1$ to $2x$ and so on. These processor rows pass data in both directions, activations upwards and errors downwards. Each such communication requires a superstep, which we will call a *big superstep*. There is approximately one big superstep per exemplar, complicated slightly by the need to fill the pipeline at the beginning and empty it at the end. The actual number of big supersteps is $(E-1) + 2(L/x - 1)$.

Consider a processor during a big superstep. The activations it has just received at the beginning of the superstep are sufficient to compute the weight corrections of its first layer of neurons. However, it cannot compute those of subsequent layers until it gets the activations from the first layer of neurons in the other processors of its row. This horizontal communication also requires supersteps, which we will call *small supersteps*. The number of small supersteps required within each big superstep is $x-1$, and each processor receives $N-y$ activation values during each one.

At the beginning of each big superstep, each of the N/p processors in a row transmits y values to each processor in the subsequent row (and, symmetrically, transmits errors to the previous row). The communication volume exchanged during a complete big superstep is therefore

$$2(y * N/y) + (N-y)(x-1)$$

and the computation required is AW/p.

Let $z = (E-1) + 2(L/x - 1)$. Then the total BSP cost of block parallelism is:

$$[(AW/p)]\, z + [2(N) + (N-y)(x-1)]\, zg + zxl$$

It is clear from this formula that better performance is achieved by making the blocks as wide as possible, that is making y large and x small. This reduces the number of small supersteps required, reducing the horizontal communication and number of barrier synchronisations required. In the limit, when $x = L/p$ and $y = N$, block parallelism reduces to *layer parallelism* (LP), which requires fewer supersteps, since the small supersteps are no longer needed. If $z = (E-1) + 2(p-1)$ then LP has BSP cost:

$$[(AW/p)E]\, z + [2N]\, zg + zl$$

Notice that layer parallelism has an opportunity for exploiting locality, because the computation becomes a pipeline which could be mapped to a linear array of processors. This is primarily of theoretical interest, however, because multilayer perceptrons do not typically have many layers, so that opportunities for exploiting parallelism in this way are limited.

For neuron parallelism, the number of supersteps is the same as for layer parallelism, since the same data dependencies exist. The amount of computation per superstep is also

	EP	BP	NP
# of supersteps	2	zx where $z = (E-1) + 2(\frac{L}{x} - 1)$	$z = (E-1) + 2(p-1)$
total computation	$\frac{AW}{p}E + (p-1)W$	$\frac{AW}{p}z$	$\frac{AW}{p}z$
total communication	$(p-1)W$	$[2N + (N-y)(x-1)]z$	$\frac{2W}{p}z$

Figure 3: Costs for Supervised Learning ($W = N^2(L-1)$).

the same, since each processor is still doing a proportionate share of the weight and activation computation, but for a random selection of the neurons. The communication per superstep requires each processor to send and receive a proportionate share of the W weights, so its communication volume is $2W/p$. These cost expressions are summarised in Figure 3.

We can now compare these methods. The computation time for all three methods is the same, except that exemplar parallelism must perform some post-processing. It is the communication volumes that differ. We compare exemplar parallelism, neuron parallelism, and layer parallelism, (since it has the least communication and synchronisation requirement of any block technique). The communication volumes are of order:

EP : N^2Lp
LP : $(E+p)N$
NP : $(E+p)N^2L/p$

Thus, regardless of the values of g and l, EP outperforms LP when $E > NLp$. This is likely to be true in practice, since the number of processors, p, is a finite and expensive resource. It is also necessarily true for reasonable error rates if Widrow's Rule [5, p 178] is used to match network size with training set size. Both EP and LP outperform NP. EP is an even clearer winner when the number of supersteps in taken into account, since l, the cost of a barrier synchronisation, is large on today's systems.

Figure 4 shows the predicted cost of the three different techniques as the number of exemplars increases. These data are for a 16 processor T3E, with $g = 1.7$ and $l = 751$, on a network with $N = 32$ and $L = 16$. For purposes of comparison, we illustrate on a network with an unrealistic number of layers. Even here, exemplar parallelism is better than layer parallelism for modest numbers of exemplars.

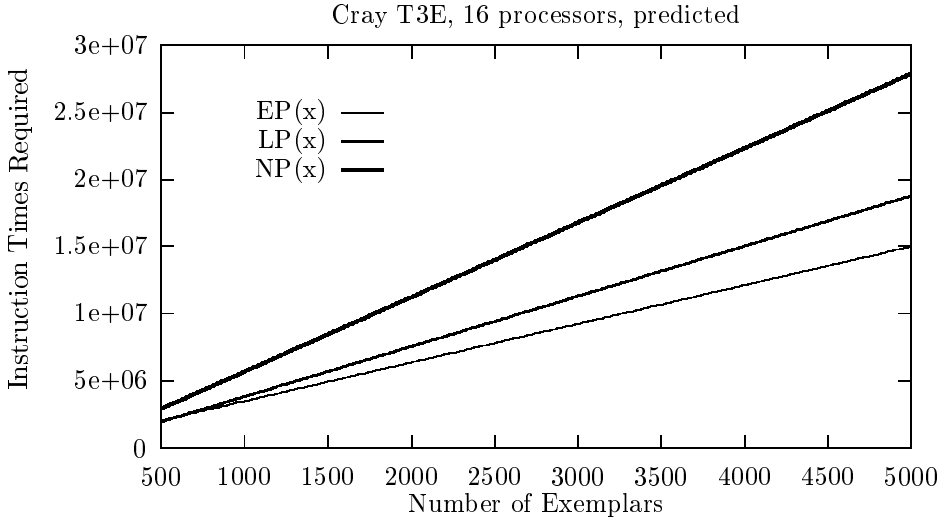

Figure 4: Predicted Cost versus Number of Exemplars – Supervised

5 Cost Analysis for Unsupervised Learning.

In this section we analyse the performance of two of the three parallelization strategies for unsupervised learning. Although the exact algorithms are different, the strategies are much the same. We analyse a competitive layer network with I inputs and J outputs. The size of a weight space is $W = IJ$, since there are IJ weighted connections between the input layer and the output layer.

Note that training a competitive layer network using a batch technique has been shown to be equivalent to k-means clustering [1] for normalised data. This justifies the use of exemplar parallelism as a parallelization technique. Block parallelism reduces to neuron parallelism, since there are only two layers, and there is no horizontal locality structure to exploit.

For exemplar parallelism, once again only two supersteps are needed: the first using a subset of the exemplars in each processor, and the second computing the final weight space. The computation for each exemplar is AW to update the accumulated weight correction, followed by $J - 1$ steps to find the local winning neuron. At the end of the first superstep, a copy of the weight updates are sent from every processor to a particular processor to do the final resolution. It receives $(p-1)W$ data and must merge these updates, an operation whose cost is $(p-1)W$.

The BSP cost of this implementation is

$$\left\lceil \frac{E}{p}(AW + J - 1) + (p-1)W \right\rceil + [(p-1)W]\, g + 2l$$

For neuron parallelism, $2E$ supersteps are needed, since each exemplar requires transmission of inputs from the input layer to each neuron, computation of activations at each neuron, and then communication of each local maximum to other neurons. The input values

	EP	NP
# of supersteps	2	$2E$
total computation	$\frac{E}{p}(AW + J - 1) + (p-1)W$	$((\frac{AW+J}{p} - 1) + (p-1) + \frac{W}{p})E$
total communication	$(p-1)W$	$(I + p - 1)E$

Figure 5: Costs for Unsupervised Learning.

for each exemplar must be passed to the output neurons at the beginning of the first phase. The computation requirement for each processor on the first phase is a proportionate part of the activation computation, AW/p, followed by $J/p - 1$ comparisons to compute the local winning neuron. The values of each of these local winners must then be communicated to all of the other processors; thus each processor receives data of size $p - 1$. In the second phase, they must then each identify the global neuron with the maximum output, requiring local computation of $p - 1$ operations, followed by updates of the neighbour neurons. Since the distribution of neurons is random, this might require updating W/p weights in the worst case (for example, an unfolded Kohonen feature map). These expressions are shown in Figure 5.

The BSP cost of this implementation is

$$\left((\frac{AW + J}{p} - 1) + (p-1) + \frac{W}{p}\right) E + (I + p - 1)\, Eg + 2El$$

EP computation cost is asymptotically smaller than that of NP when $E > W$. Once again, this is necessarily true for plausible error rates if Widrow's Rule [5] is used to choose the size of the network and training set. The communication cost of EP is roughly pW while that of neuron parallelism is roughly $(p+I)E$, so that exemplar parallelism is a clear winner, even more so if synchronisation cost is considered.

Figure 6 shows the predicted cost of these two techniques for the same processor configuration as before, the Cray T3E with 16 processors, using $I = J = 32$. This shows the advantage of EP for even small numbers of exemplars.

6 Conclusion

We have presented a BSP cost analysis of three parallelization strategies for both supervised and unsupervised neural network training. BSP's clear and accurate cost model shows

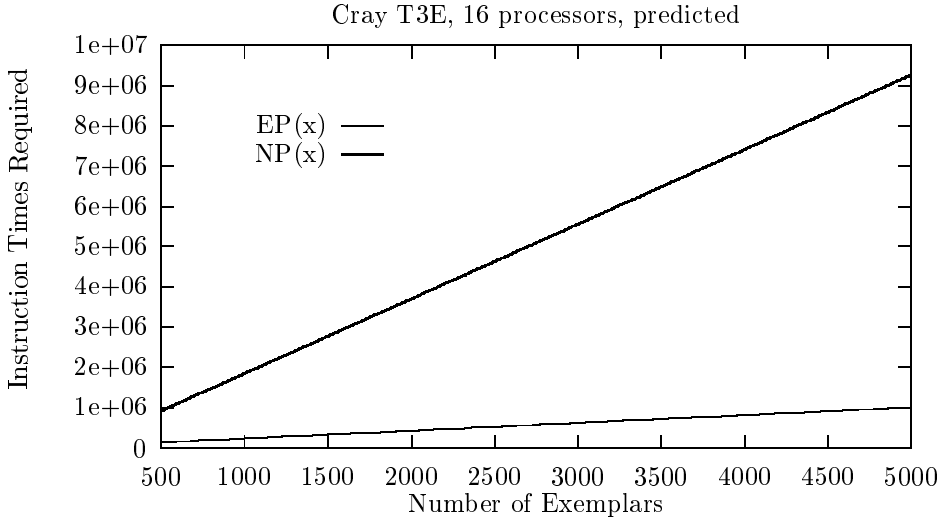

Figure 6: Predicted Cost versus Number of Exemplars – Unsupervised

definitively that exemplar parallelism is the preferred technique. This is not surprising in the sense that partitioning a neural network across the processors of a parallel computer necessarily generates heavy communication. We have quantified the extent to which this is true within a framework that captures critical architecture properties. Thus there is no need for extensive simulations to discover which techniques are not appropriate.

References

[1] S. Becker and M. Plumbley. Unsupervised neural network learning procedures for feature extraction and classification. *Journal of Applied Intelligence*, 6:1–21, 1996.

[2] M. Besch and H.W. Pohl. How to simulate artificial neural networks on large scale parallel computers exploiting data parallelism and object orientation. Technical Report TR-94022, GMD FIRST Real World Computing Laboratory, November 1994.

[3] M. Besch and H.W. Pohl. Flexible data parallel training of neural networks using MIMD computers. In *Third Euromicro Workshop on Parallel and Distributed Processing*, January 1995.

[4] P. Frasconi, M. Gori, and G. Soda. Daphne: Data parallelism neural network simulator. *International Journal of Modern Physics C*, 1992.

[5] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1994.

[6] J.M.D. Hill and D.B. Skillicorn. Practical barrier synchronisation. Technical Report TR-96-16, Oxford University Computing Laboratory, October 1996.

[7] Jonathan M. D. Hill, Paul I Crumpton, and David A. Burgess. The theory, practice, and a tool for BSP performance prediction. In *EuroPar'96*, volume 1124 of *LNCS*, pages 697–705. Springer-Verlag, August 1996.

[8] A. Hyaric. Converting the NAS benchmarks from MPI to BSP. High Performance Computing and Networks '96 Presentation, `http://merry.comlab.ox.ac.uk/oucl/users/hyaric/doc/BSP/NASfromMPItoBSP/`, 1996.

[9] M. Nibhanupudi, C. Norton, and B. Szymanski. Plasma simulation on networks of workstations using the bulk synchronous parallel model. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Athens, GA, November 1995.

[10] T. Nordström. Designing parallel computers for self organizing maps. In *Proceedings of the 4th Swedish Workshop on Computer Systems Architecture*, 1992.

[11] D.A. Pomerleau, G.L. Gusciora, D.L. Touretzky, and H.T. Kung. Neural network simulation at Warp speed: How we got 17 million connections per second. In *IEEE International Conference on Neural Networks*, July 1988.

[12] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. Questions and answers about BSP. *Scientific Programming*, to appear. Also appears as Oxford University Computing Laboratory, Technical Report TR-15-96, November 1996.

[13] N.B. Šerbedžija. Simulating artificial neural networks on parallel architectures. *Computer*, 29, No.3:56–63, 1996.

[14] M. Whitbrock and M. Zagha. An implementation of backpropagation learning on GF11, a large SIMD parallel computer. *Parallel Computing*, 14:329–346, 1990.

[15] X. Zhang, M. McKenna, J.P. Mesirov, and D.L. Waltz. The backpropagation algorithm on grid and hypercube architectures. *Parallel Computing*, 14:317–327, 1990.