

Using the BSP Cost Model to Optimise Parallel Neural Network Training

R.O. Rogers and D.B. Skillicorn

Department of Computing and Information Science
Queen's University, Kingston Canada

Abstract. We derive cost formulae for three different parallelisation techniques for training supervised networks. These formulae are parameterised by properties of the target computer architecture. It is therefore possible to decide the best match between parallel computer and training technique. One technique, exemplar parallelism, is far superior for almost all parallel computer architectures. Formulae also take into account optimal batch learning as the overall training approach.

1 Introduction

Neural network learning is expensive, and hence a natural application for parallelism [1, 2, 8]. Almost all of these papers demonstrate that speedup can be achieved, although it is often disappointing when compared to the resources used. In this paper we answer a much more interesting question: given a range of parallel computers, and a set of parallelisation strategies for training, which combination delivers the best performance? Knowing the answer makes it possible to match parallelisation strategy to architectures, but also makes it possible to match architecture choice and number of processors to the particular problem domain of interest. One technique, exemplar parallelism, is much superior to all of the others over almost all hardware configurations and all but the smallest training data set sizes. Thus there is little point investigating more esoteric parallelisation strategies. We derive these results using the Bulk Synchronous Parallelism (BSP) cost model, which is both simple and accurate on real parallel computers. Although derived for a particular computation model, these results are generally applicable.

Section 2 describes the properties of BSP and its cost model. Section 3 describes the parallelisation strategies that are investigated. Section 4 derives the costs of training for each of these parallelisation strategies. Section 5 reviews the training performance improvements possible by the use of batch learning. Section 6 combines these performance results to give cost formulae from which optimal strategies can be derived for individual architectures.

2 Bulk Synchronous Parallelism

Bulk Synchronous Parallelism (BSP) is a parallel computation model that facilitates the development and analysis of general-purpose parallel software [5, 9]. A

BSP abstract machine is very simple: it is a set of processor-memory pairs, linked by some interconnection network. This abstraction can be easily implemented by any MIMD machine. BSP programs consist of sequences of *supersteps*. Supersteps specify the actions of every processor in the architecture, that is they are global actions that fill the entire computer. Each superstep consists of three sequential phases: computation in each processor, using only values in its local memory; communication between processors; and a barrier synchronisation, after which communicated values become visible in the local memories of the processors to which they were transferred. This structure separates data transfer from synchronisation and makes it impossible to write programs that deadlock. The current best implementation of the BSP model is *BSPLib*, a library callable from C and Fortran [3].

The cost of a BSP superstep can be straightforwardly computed from the program text and two architecture-specific parameters. The first of these parameters, g , measures the permeability of the parallel machine to uniformly-random traffic [9], in units of flops per transmitted word. The second parameter, l , measures the average time required for a barrier synchronisation, in units of flops.

The cost of a BSP superstep is the sum of the cost of each phase:

$$\text{MAX}_{\text{processes}} c_i + \text{MAX}_{\text{processes}} h_i g + l \quad (1)$$

where, for processor i , c_i is the number of floating point instructions performed during the local computation phase, and h_i is the number of words sent or received. The cost of the program is the sum of the cost of each superstep.

The BSP cost model has been shown to be highly accurate for estimating the execution times of real-world parallel applications – typically within a few percent of the actual value [5].

3 Parallelisation Strategies

Different parallelisation strategies for neural networks can be categorised in terms of their granularity. We consider three levels of parallelism:

Exemplar Parallelism (EP). The data set is partitioned among each of the processors in the parallel computer. Each processor receives an identical copy of the initial neural network, and calculates the error gradients for its exemplars (in the standard sequential fashion). At the end of an epoch, a total exchange of gradient estimates takes place (hence this technique is *deterministic*). Each processor then updates its weights, leaving every processor with an identical copy of the network. In effect, each network is in the state that it would have been in if it had trained on the entire data set.

Block Parallelism (BP). The neural network is divided into blocks of adjacent neurons and each is allocated to a processor. For simplicity, we assume that these blocks are non-overlapping and rectangular, with depth x and width y .

This approach to parallelisation attempts to take advantage of the locality that exists between adjacent neurons.

Neuron Parallelism (NP). Every neuron is treated as a parallel process. No attempt is made to exploit the locality between neurons – they are randomly allocated to processors.

There are two other levels of neural parallelism that have been considered: training-session parallelism, and weight parallelism. The results here imply that neither extreme is attractive.

4 Cost Analysis using BSP

The neural network that will be considered in these cost analyses is the cross-entropy multi-layer perceptron. Despite the use of a specific network, the results apply to a wide range of supervised neural network algorithms. Detailed derivation of these cost formulae can be found in [6].

We assume that the neural network consists of L layers with M neurons per layer, with each layer fully connected to the neurons in the preceding and succeeding layers. The total number of neurons is LM , and the total number of weights is $W = LM^2$. The rectangular structure of this hypothetical network is a superset of all other regular network topologies. We assume N exemplars in the training set, and W adjustable weights.

Training the network for a single epoch requires evaluating the network function and calculating the error gradient for each exemplar, and updating every weight at the end of the epoch. All three phases require a constant number of computations per weight, so the computation cost of training for an epoch is:

$$C_E = ANW \tag{2}$$

and A is determined empirically. Note that this cost is independent of the network topology, except as it affects the number of weights.

Exemplar Parallelism Cost Analysis. The cost of network training is reduced by partitioning the data set across the processors. Each processor receives N/p examples, where p is the number of processors.

Two supersteps are required. The first computes the weight adjustments for each processor and distributes them globally, while the second combines the adjustments received from each processor and updates the network weights. The computational cost of the first superstep is $A\frac{N}{p}W$ from Equation (2). Once each processor has derived the error gradients using its local data set, these results are exchanged globally. The cost of the communication phase of this superstep is $(p-1)Wg$.

The second superstep involves combining the W gradient estimates received by each processor to update the network weights. The computational cost of combining these estimates is $(p-1)W$. No communication needs to be performed during this superstep.

The total cost of exemplar parallelism for each epoch is:

$$C_{EP} = \left[\frac{ANW}{p} + (p-1)W \right] + [(p-1)W]g + 2l \quad (3)$$

Block Parallelism Cost Analysis. The network is partitioned into non-overlapping rectangular blocks of neurons which are distributed to each of the processors of the parallel computer. Clearly $xyp = LM$. The total number of steps required to process a data set of N exemplars is $(N-1) + 2(\frac{L}{x} - 1)$, because of pipelining. Each of these steps will be called a *big superstep*.

The cost of the computation phase of each big superstep is simply the forward and backward propagation costs for each of the xy neurons in the block. Thus, the cost of computation is Axy , which can be rearranged, using the equality $xyp = W$, to AW/p . The cost of communication comes from sending the outputs of the first and last layer in the block to the processors containing adjacent layers. Thus, $2y$ values are sent to the M/y processors in the two adjacent columns. The cost of the communication phase for each big superstep is therefore $2yM/y$.

However, if each block spans multiple network layers ($x > 1$), additional communication is needed between distinct processors containing neurons from the same layer. Because each processor only contains y neurons from each layer, the other $M-y$ values are required. This additional set of communications is called a *small superstep*. Because there are x layers in each block, $(x-1)$ small supersteps are required for each big superstep. The communication cost for small supersteps is therefore $(M-y)(x-1)$. The complete BSP cost for block parallelism is:

$$C_{BP} = (N-1) + 2\left(\frac{L}{x} - 1\right) \left[\frac{AW}{p} + (2M + (M-y)(x-1))g + xl \right] \quad (4)$$

Clearly, block parallelism can be made more efficient by reducing the number of small supersteps. If each processor is assigned an entire layer, the small supersteps can be eliminated. This is known as layer parallelism. The BSP cost reduces to:

$$C_{LP} = (N-1) + 2(L-1) \left[\frac{AW}{p} + 2Mg + l \right] \quad (5)$$

Neuron Parallelism Cost Analysis. An equal number of neurons, ML/p , is assigned to each processor. Neuron parallelism performs $(N-1) + 2(L-1)$ supersteps per epoch.

The computation cost is distributed evenly among the processors in the parallel machine so that each performs $\frac{ML}{p}$ of the total computations. The total cost of the local computation phase for each superstep is AW/p .

The cost of the communication phase is greater for neuron parallelism than for block parallelism because of the absence of locality. In the worst case, a neuron must transmit its output to all p processors. The cost of communication is therefore $2M$. Because this cost is shared by all of the ML/p neurons in each

processor, the total communication cost for each superstep is $2M^2L/p$. The BSP cost of neuron parallelism is:

$$C_{NP} = (N - 1) + 2(L - 1) \left[\frac{AW}{p} + \frac{2WL}{p}g + l \right] \quad (6)$$

	EP	LP	NP
total computation	$\frac{N}{p}AW$	$N\frac{AW}{p}$	$N\frac{AW}{p}$
total communication	$W(p - 1)g$	$N2Mg$	$N\frac{2W}{p}g$
total synchronisation	$2l$	Nl	Nl

Table 1. Parallel implementation costs per epoch.

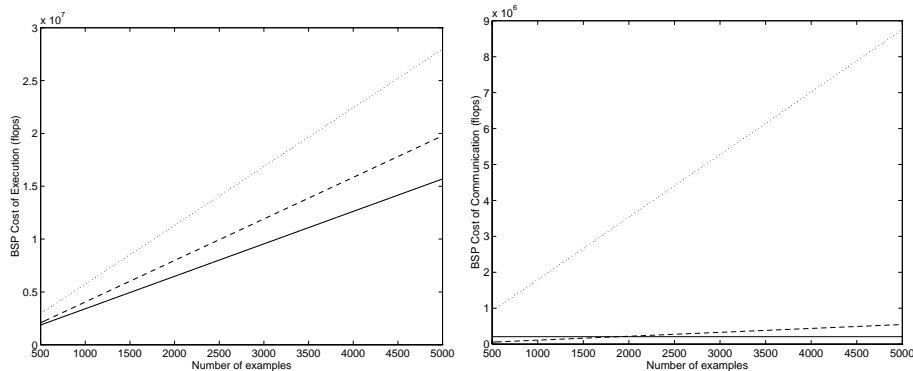
Comparison of Parallelisation Strategies. There are some practical constraints on the range of values of the BSP parameters, which simplifies comparisons. For example, because processors are a finite and expensive resource, the number of processors is typically small relative to the size of the network, in particular $p \leq ML$. An additional constraint is the size of the network in relation to the number of data examples. As a rule of thumb, the size of the data set should be proportionally greater than the number of weights by some constant factor ϵ [4, p. 179],

$$N \gg \frac{W}{\epsilon} \quad (7)$$

where ϵ is inversely proportional to the desired accuracy of the network. Therefore, the number of examples N is likely to be considerably larger than the number of weights, especially for data mining applications. Simplified costs are compared in Table 1.

The main difference between the three parallelisation strategies is the amount of communication required and the number of barrier synchronisations performed. The superiority of layer parallelism over neuron parallelism is apparent. For layer and exemplar parallelism, it is not so obvious which technique is superior. Exemplar parallelism performs less communication than layer parallelism if $N > \frac{1}{2}ML(p - 1)$. Rule of thumb (7) suggests that this is usually true. Therefore, exemplar parallelism is the optimal technique for parallelising training in supervised neural networks.

Figure 1 shows the predicted parallel implementation costs of a supervised neural network using the three methods of parallelisation as a function of N . These results are for a neural network with 16 layers of 32 neurons, on a 16-processor Cray T3E parallel computer with $g = 1.7$ and $l = 751$. The constant A is assumed to be 6. For the purpose of comparison, an unrealistically large



(a) Predicted total costs.

(b) Predicted communication costs.

Fig. 1. Comparison of predicted costs for exemplar, layer and neuron parallelism. Solid line – exemplar parallelism, dashed line – layer parallelism, dotted line – neuron parallelism.

number of layers is used in this example. Even so, exemplar parallelism is better than layer parallelism for modest numbers of examples. Figure 1 also compares the communication costs of the three parallelisation strategies. It is clear that, even for a very small number of examples and a very large network, the cost of communication for layer parallelism quickly outgrows the communication required by exemplar parallelism.

5 Batch Learning

A neural network algorithm that uses every example to compute the gradient (deterministic learning) reaches the best available approximation of the true error gradient. Batch learning compromises the accuracy of the gradient estimate to achieve greater speed of convergence. The training data set is partitioned into disjoint, equal-sized batches. The network updates its weights only after evaluating the gradient for all the exemplars in a batch. The gradient for a batch can be viewed as a noisy estimate of the deterministic gradient. If the noise term is sufficiently small, the batch delta weights are close to the deterministic delta weights, and the network is in almost the same state after training on the batch as after training on the entire data set. Using smaller batches speeds convergence as long as the batch gradient continues to approximate the deterministic gradient. Unless the database is completely redundant, there is eventually insufficient information in a small set of examples to accurately estimate the true gradient. A tradeoff exists between the accuracy of the gradient estimate, and the number of weight updates per epoch. Between the extremes of deterministic and stochastic learning there is an optimal batch size that maximises the convergence speed of the network.

The number of epochs required to train batch learning networks behaves as shown in Figure 2. The curve is divided into two regions. For batch sizes from

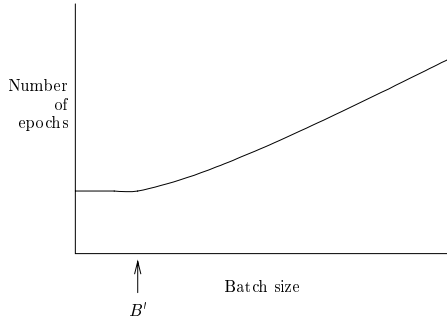


Fig. 2. Number of epochs required for convergence as a function of batch size.

B' to N , the number of epochs required for convergence increases linearly with the batch size. For batch sizes smaller than B' , the number of epochs required for convergence is constant as B decreases. Details can be found in [7].

6 Using Parallelism and Batch Learning

For sequential computation, batch learning provides substantial performance improvement. This is not so clear for a parallel implementation because the multiple weight updates per epoch require additional communication and synchronisation. Combining the batch learning network training results with the BSP cost model results enables us to determine this definitively.

The BSP cost formulae need to be slightly altered to take batch learning into account. Exemplar parallelism is the superior strategy for parallelising neural networks, so only this technique is considered. For batch learning using b batches, b weight updates are performed each epoch, and the number of super-steps increases by a factor of b .

The BSP cost of exemplar parallelism per epoch for batch learning neural networks is:

$$C_{EP}^{\text{batch}} = b \left[\frac{N}{b} \frac{AW}{p} + (p-1)Wg + 2l \right] \quad (8)$$

and so the total cost for training for E epochs is

$$C_{EP}^{\text{total}} = E \left[\frac{NAW}{p} + b(p-1)Wg + 2bl \right] \quad (9)$$

Consider batch sizes in the range where number of epochs required for convergence depends linearly on B . In this range, we can write $E = c/b$, for some constant c . Applying this to (9), we get:

$$C_{EP}^{\text{total}} = \frac{c}{b} \frac{NAW}{p} + c(p-1)Wg + 2cl \quad (10)$$

The cost of communication and synchronisation are independent of the number of batches. Thus, minimising the overall cost requires minimising the computation term. This happens when b is as large as possible, that is when the batch

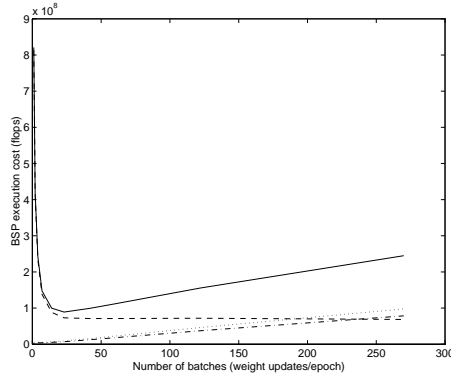


Fig. 3. Breakdown of the BSP cost for training on the thyroid database using a 4-processor IBM SP2. Solid line – total BSP cost of parallel implementation, dashed line – cost of computation, dot-dashed line – cost of communication, dotted line – cost of synchronisation.

size, B , is as small as possible consistent with being in the appropriate range. This occurs when $B = B'$ or $b = b' = N/B'$.

When batch sizes are smaller than B' , the number of epochs required is approximately constant. Thus the total cost is

$$C_{EP}^{\text{total}} = C \left[\frac{NAW}{p} + b(p-1)Wg + 2bl \right] \quad (11)$$

for some constant, C . Now the computation term is constant, but the communication and synchronisation costs increase with the number of batches (or with decreasing batch size). Clearly this is minimised by choosing as few batches as possible, that is once again choosing $B = B'$. The shape of this graph for a real data set is shown in Figure 3 using the communication parameters of a 4-processor IBM SP2.

Note that p appears in the denominator of the computation term but linearly in the communication term. Thus for small values of p and large (i.e. poor) values of g , parallelism may not increase performance.

Although these performance formulae are expressed in terms of the BSP architecture parameters, the conclusions do not change for other computational frameworks. For example, implementing without barrier synchronisations is equivalent to setting $l = 0$, which changes the absolute, but not the relative, performance of different parallelisation techniques. The g parameter captures a fundamental property of architectures rather than programming models, but even for an idealised machine with $g = 1$ the relative performance results change little – the crossover between exemplar parallelism and layer parallelism moves to the right (but this is not practically interesting for networks with few layers).

7 Conclusions

We have constructed parallelised algorithms for training supervised networks in the BSP style. From these algorithms, predicted execution costs based on architectural parameters have been derived. We conclude that exemplar parallelism is the technique of choice for all architectures, and all but the smallest training data sets. We then derive formulae for training cost using both exemplar parallelism and batch learning.

These results show that there is little to be gained by trying to exploit the fine-grained parallelism available in neural network computation, because to do so requires too much extra communication. It also suggests constraints on specialised hardware if it is to provide any real increase in training performance.

References

1. D. Anguita, A. Da Canal, W. Da Canal, A. Falcone, and A.M. Scapolla. On the distributed implementation of the back-propagation. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN'94)*, pages 1376–1379, Sorrento, Italy, 1996.
2. 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*. San Remo, Italy, 1995.
3. M.W. Goudreau, J.M.D. Hill, K. Lang, W.F. McColl, S.D. Rao, D.C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for a BSP Worldwide standard. BSP Worldwide, <http://www.bsp-worldwide.org/>, April 1996.
4. S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey, 1994.
5. J.M.D. Hill, P.I. Crumpton, and D.A. Burgess. Theory, practice, and a tool for BSP performance prediction. In *Europar'96*, volume 1124 of *LNCS*, pages 697–705. Springer-Verlag, 1996.
6. R.O. Rogers. A framework for parallel data mining using neural networks. Technical Report 97-413, Queen's University, Department of Computing and Information Science, November 1997.
7. R.O. Rogers and D.B. Skillicorn. Batch size and training times in supervised and unsupervised networks, December 1997.
8. N. Serbedzija. Simulating artificial neural networks on parallel architectures. In *Computer*, volume 29, pages 53–63, 1996.
9. 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.