Implementation of MLP Networks Running Backpropagation on Various Parallel Computer Hardware Using MPI

Tobias Czauderna[1] and Udo Seiffert[1,2]

[1] Leibniz-Institute of Plant Genetics and Crop Plant Research
Pattern Recognition Group
Corrensstraße 3, 06466 Gatersleben, Germany

[2] University of Magdeburg
Institute of Electronics, Signal Processing and Communications
P.O. Box. 4120, 39016 Magdeburg, Germany
e-mail: {czauderna, seiffert}@ipk-gatersleben.de

Abstract: *Multiple-Layer Perceptrons (MLPs) running Backpropagation are still a very frequently utilized artificial neural networks paradigm. Particularly when applied to very high-dimensional data sets leading to rather large networks, their training may take too long. Besides some architectural or solely numerical modifications, especially a parallel implementation is suitable to speed up the network training.*
This paper evaluates a Message Passing Interface (MPI) based parallel variant of Backpropagation which was run on a number of different parallel computer architectures. A standard character recognition problem with a wavelet transform based feature data set of 262 *input dimensions is applied as benchmark environment.*

Keywords: Parallel artificial neural network, Multiple-Layer Perceptron (MLP), Backpropagation, Message Passing Interface (MPI), Symmetrical Multi-Processor (SMP), Beowulf cluster.

1 Introduction

The brain consists of tens of billions of neurons densely interconnected. Its complexity and efficiency is only possible because many natural processing steps generally run in parallel – the neurons generally run in parallel. Since Artificial Neural Networks (ANNs) aim at a more or less close technical implementation of the neurobiological principles to provide an adaptive, trainable and biologically motivated tool for a large variety of complex problems, a parallel implementation is inherently feasible [1].

But not only from the scientific point of view artificial neural networks being run in parallel are highly appreciated or even required. Despite a continuously increasing speed of microprocessors (not only the sheer clock rates), as soon as the tasks, ANNs are applied to, become more complex, the computation time rapidly exceeds all tolerable limits. A promising – and moreover easier to reach than expected – loophole is a parallel implementation of the ANN [11, 13].

When looking at existing equipment in standard computer labs or making plans to acquire dedicated potentially parallel hardware, the question arises, which architecture is suitable for a particular problem. The answer depends on a number of factors, i.e.

- the neural network topology (size; layout of layers),
- the neural network training and control strategy (required numerical accuracy, effort and complexity; algorithmic structures; data transfer within the net),
- utilized processors and their interconnectivity (clock rate; cache size; memory access),
- programming model and language, software libraries.

A recent survey of the state-of-the-art can be found in [13]. According to the above mentioned list this paper is focussed on details of a versatile and high performance implementation of Multiple-Layer Perceptrons trained with Backpropagation on various parallel hardware. After a comprehensive description of the test environment in the next section, a close look at the results accompanied by an in-depth discussion is provided in Sect. 3.

2 Test Environment

This section describes the used test bed in terms of the applied neural network, the data set as well as the hardware and software systems.

2.1 Artificial Neural Network

In the field of supervised trained neural nets Multiple-Layer Perceptrons (MLPs) [2] are still very frequently used. Although there are so many extensions and modifications and the advantages of other neural network paradigms have often been demonstrated, MLPs still offer a reasonable solution for many problems in a fast and easily manageable way.

In general, parallel processing can be achieved in different ways – data set parallelism [12] and algorithmic parallelism [8, 6]. While the first method can be easily applied simply by partitioning the data set, the latter method is much more sophisticated and requires a deeper understanding of the underlying algorithm. As a prerequisite of this method the algorithm itself must be generally parallelizable [3]. However, as stated in the introductory section, this is always guaranteed for neural nets due to their inherent natural parallelism.

Algorithmic parallelism offers two possible schemes for the MLP: (1) mapping each layer of the network to a processor and (2) mapping a block of neurons to a processor. A special case of the second scheme is to map each neuron of the network to a processor so that the parallel computer becomes a physical model of the network. The first scheme limits the parallelism since most MLPs consist of only a few layers (one input layer, one to three hidden layers and one output layer). Moreover, the processor load balancing can become disadvantageous, since the number of neurons per layer can be different and thus the computational load per processor can be different. A parallel algorithm for this scheme has to have a pipeline structure, due to the fact that the layers can not be processed independently.

The implementation described in this paper uses the second scheme. The neurons of the network are dynamically mapped to the processors, the actual topological distribution depends on the number of available processors and is set up at run-time. This kind of parallelism has a significantly high expense of communication for the exchange of data between the processors [13]. Thus, it was necessary to optimize the implementation regarding the expense of communication to get a reasonable performance of the parallel implementation compared to a sequential implementation (see Sect. 3). The Message Passing Interface offers some promising possibilities to optimize the communication (e.g. collective communication instead of point-to-point communication) [10].

The parallelism of the implementation is constrained to the training phase since it is the most time consuming part. Furthermore the implementation offers an unlimited number of neurons per layer (up to three hidden layers) and an unlimited amount of data. For these attributes the only constraint is given by the size of the memory of the parallel computer. The two most frequently used transfer functions (sigmoid and hyperbolic tangent) can be used.

2.2 Data Set

The data set was obtained from the *FL3 Handwritten Symbol Database* which is a subset of the *NIST Special Database 1* of the National Institute of Standards and Technology of the USA [9]. It consists of 3471 images (32×32 pixel) which represent the digits $0 \ldots 9$. It was randomly split into 2280 training examples and 1191 test examples. 262 features were extracted from every image based on a wavelet transform (courtesy Th. Villmann, University Leipzig). Since this data set is solely utilized to set up a benchmark environment and this paper does not address the recognition problem of the data set itself, the recognition performance is not further relevant. One solution of the recognition problem of this data set using Learning Vector Quantization (LVQ) networks is presented in [14].

2.3 Parallel Hardware Systems

We were able to use a number of different parallel computer architectures for the benchmark tests. Table 1 shows a survey of the system specifications. Since the exchange of data between the processors respectively the nodes was realized through message passing, in addition to the system specifications two more criteria are important to yield a reasonable performance: the point-to-point communication bandwidth and point-to-point communication latency [4, 10] which are displayed in Fig. 1 for the used systems.

The hardware test bed covers a wide range of available parallel computers with three Symmetrical Multi-Processor (SMP) machines and two Beowulf clusters and four different types of processor respectively node interconnection topologies (crossbar switch, 2d-torus topology for the SMPs as well as Myrinet, Fast Ethernet for the clusters).

Table 1: Survey of the used parallel computer architectures giving the most significant hardware specific information, operating system and parallel programming environment.

System	System Specifications
Sun V880	8 UltraSparc III+ processors (1,2 GHz)
	8 MB cache per processor, 16 GB memory
	crossbar switch
	Solaris 9, SUN MPI 6.0
HP 9000 Superdome	64 PA-8700 RISC processors (750 MHz)
	2,25 MB Cache per processor, 128 GB memory
	crossbar switch
	HP-UX 11.0, HP MPI 1.08
HP GS1280	32 Alpha EV7 processors (1,15 GHz)
	1,75 MB cache per processor, 128 GB memory
	2d-torus topology
	Compaq Tru64 UNIX 5.1B, Compaq MPI 1.9
Beowulf Cluster	10 nodes each with two Pentium IV Xeon processors (3,06 GHz)
Pentium IV	512 kB cache per processor, 1 GB memory per node
	Myrinet, Fast Ethernet
	Red Hat 7.1, Myrinet MPICH 1.2.5..10, Fast Ethernet MPICH 1.2.5.2
Beowulf Cluster	32 nodes each with two Pentium III Tualatin processors (1,26 GHz)
Pentium III	512 kB cache per processor, 1 GB memory per node
	Fast Ethernet
	Red Hat 7.1, MPICH 1.2.5.2

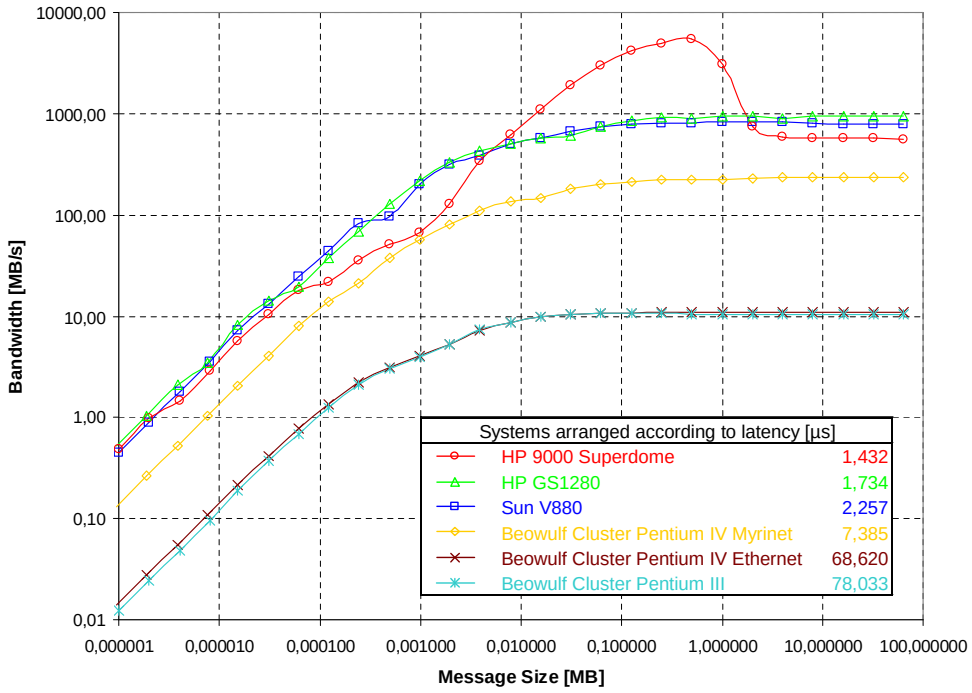

Figure 1: Point-to-point communication bandwidth and point-to-point communication latency (inlet) of the used parallel computer architectures.

2.4 Software

Many parallel implementations of neural nets are designed system dependently regarding the underlying hardware/software [7]. Generally a more versatile implementation can be advantageously run on different platforms without any source code modifications. This comparative study is based on MPI because it has developed into the defacto standard for message passing paradigms which are available on almost all parallel computer architectures. The described MLP implementation was achieved using the C programming language with the language extensions of MPI. Another well-known message passing system is the Parallel Virtual Machine (PVM) [5].

It was necessary to use a message passing paradigm since two of the used parallel computer architectures are Beowulf clusters with distributed memory (see Table 1). With the only use of shared memory systems, OpenMP [5] could have been used for the parallelization as well.

3 Results and Discussion

The implementation was evaluated regarding the performance on different parallel computers (see Sect. 2.3). To compare the performance of all parallel computers, the training time for the benchmark data set was measured running the implementation on one to eight processors. The parallelism is constrained as usual to the training phase (see Sect. 2.1). Fig. 2 shows the results. In order to achieve numerical consistency for the evaluation, all training parameters, including a fixed iteration depth (30 epochs), have been kept constant. The sigmoid function was used as transfer function.

Generally Fig. 2 shows two different results. The implementation running on one to eight processors on the three SMPs and the Beowulf Cluster Pentium IV with Myrinet scales very well. In contrast, the Beowulf clusters with Fast Ethernet are not suitable for MLP networks.

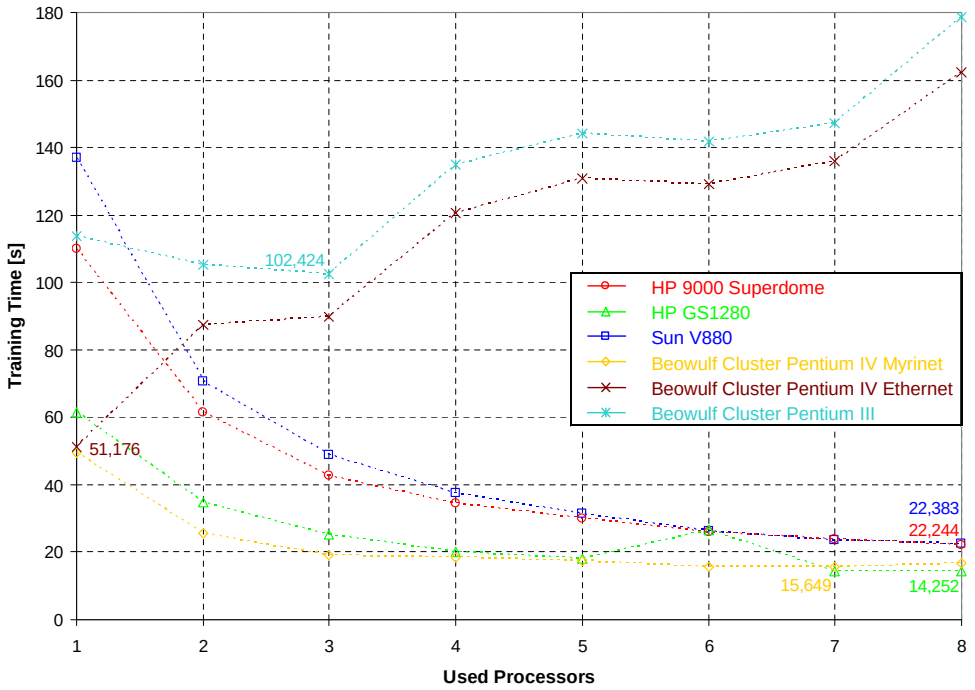

Figure 2: Training time for the benchmark data set running the implementation on one to eight processors on the used parallel computer architectures.

The parallel execution of the training phase is only reasonable when the gain in computing time can at least compensate the expense of communication time. Obviously this can not be achieved for all of the used parallel computers. Referring to Sect. 2.3, from the systems point of view the impact of communication mainly depends on two criteria: the bandwidth and the latency of the underlying processor respectively node interconnection topology. Since the messages are rather small (typical size is smaller than 8 kB), the most significant attribute is the latency (see Fig. 1). For the Beowulf clusters with Fast Ethernet this value is considerably higher than for the three SMP machines and the Beowulf Cluster Pentium IV with Myrinet. Therefore the gain in computing time can not compensate the expense of communication for these two architectures.

The smallest obtained training time vs. the number of used processors for all tested parallel computers is shown in Fig. 2. For the three SMP machines this was obtained using the maximum of eight processors and for the Beowulf Cluster Pentium IV with Myrinet already at seven processors. The results for the Beowulf Cluster Pentium III with Fast Ethernet show the minimum training time for the use of three processors. This time is not further relevant since it is inconsiderably smaller than the training time for the use of one processor on the same machine. A comparison of the results for the two Beowulf clusters with Fast Ethernet for the use of three or more processors shows that the results only depend on the node interconnection topology and become independent from the type of the used processor itself.

4 Conclusions

This paper evaluated the suitability of a number of different parallel computer systems by means of a Message Passing Interface (MPI) based Multiple Layer Perceptron/Backpropagation implementation. It is demonstrated that neural networks can be advantageously implemented on parallel computers with a significant speed-up. Due to their sophisticated system architecture, SMP machines are very suitable for neural network implementations, whereas the class of the significantly less expensive computer clusters requires a price rising fast interconnection network.

Acknowledgements

The authors would like to express their gratitude to Thomas Villmann for providing the benchmark data set as well as to Rolf Knocke and Jörg Schulenburg for their administrative and technical support. Funding for this work was provided by two grants of the German Federal ministry of Education and Research (BMBF), No. 0312706A and No. 0313115.

References

[1] M. A. Arbib. Artificial intelligence and brain theory. *Ann. Biomed. Eng.*, 3:238–274, 1975.

[2] Y. Chauvin and D. E. Rumelhart, editors. *Backpropagation: Theory, Architectures, and Applications*. Developments in Connectionist Theory. Lawrence Erlbaum, Mahwah, NJ, 1995.

[3] R. Greenlaw, H. Hoover, and W. Ruzzo. *Limits to Parallel Computation*. Oxford University Press, Oxford, UK, 1995.

[4] W. Gropp and E. Lusk. Some early performance results with MPI on the IBM SP1 - early draft. 1995. (available at http://www.psc.edu/general/software/packages/mpich/perf.ps).

[5] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message passing interface*. The MIT Press, Cambridge, Ma., 1999.

[6] V. Kumar, S. Shekhar, and M. B. Amin. Scalable parallel formulation of the backpropagation algorithm for hypercubes and related architectures. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1073–1090, 1994.

[7] M. Misra. Parallel environments for implementing neural networks. *Neural Computing Surveys*, 1:48–60, 1997.

[8] M. Misra and V. K. P. Kumar. Implementation of neural networks on parallel architectures. In B. Souček, editor, *Fast Learning and Invariant Object Recognition*. John Wiley and Sons, Inc., 1992.

[9] National Institute of Standards and Technology. FL3 handwritten symbol database - subset of the NIST special database 1. NIST. ftp://sequoyah.ncsl.nist.gov/pub/databases/data.

[10] P. S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc., San Francisco, Ca., 1997.

[11] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. The MIT Press / Bradford Book, Cambridge, Ma., 1986.

[12] P. Saratchandran, N. Sundararajan, and S. K. Foo, editors. *Parallel Implementations of Backpropagation Neural Networks on Transputers: A Study of Training Set Parallelism*. Number 3 in Progress in Neural Processing. World Scientific Pub, Singapore, 1996.

[13] U. Seiffert. Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, 57:135–150, March 2004.

[14] T. Villmann, F.-M. Schleif, and B. Hammer. Supervised neural gas and relevance learning in learning vector quantization. In *Proceedings of the Workshop on Self-Organizing Maps (WSOM'03)*, pages 47–52, Kitakyushu, Japan, 2003.