Implementation of the Backpropagation Neural Network Learning Algorithm on the Quadrics Supercomputer Family

Sergio Taraglio
ENEA, INN - RIN, C.R. Casaccia, Via Anguillarese 301, 00060 Roma

Abstract

A parallel implementation of a software library to build and train Multi Layer Perceptrons via the Back Propagation algorithm is presented. The target machine is the SIMD massively parallel supercomputer Quadrics. Performance measures are provided on three different machines with different number of processors. An applicative test case, concerning the defect classification in rolled steel strips, is provided.

Introduction

In the beginning of their history, artificial neural networks architectures were employed to mimic in an extremely simplified way the nervous system. Afterwards a more engineering point of view entered the scene. Nowadays the use of such computing technique is spread among a huge and increasing variety of fields of application.

We present here the implementation of the most classical learning algorithm for the most popular neural architecture: i.e. the backpropagation used to train a multi layer perceptron, see [1].

The target computer used for this aim is the Quadrics super computer, a SIMD massively parallel machine derived from the APE 100 project of the Italian National Institute for Nuclear Physics (INFN).

The parallel algorithm is here presented together with the performance evaluation on the Quadrics super computer family. A field test is indicated for an evaluation of performances on a real application. The application is represented by a classifier of defects in the steel industry, a project developed under the UE Parallel Computing Initiative called CAPRI in the ESPRIT framework, for more details see [2].

The Quadrics Architecture

The Quadrics parallel supercomputer series derive from the INFN project APE 100, which was born to solve the need for highly intensive computing in the field of particle physics, namely Quantum Chromo Dynamics (QCD) see [3] and [4].

It is a SIMD (Single Instruction Multiple Data) parallel machine, attached to a host computer which is responsible for the uploading/downloading of programs and data and the managing of I/O operations.

Quadrics is composed of an integer CPU (Z-CPU) which controls the flux of the program and the integer arithmetic and of an array of custom floating point units (MAD, Multiply and Add Device), each one equipped with its own memory. The nodes are logically arranged on the vertexes of a 3D cubic mesh and are capable of synchronous, parallel first neighbours communications.

In the Quadrics architecture, each processor board contains eight floating point units. The different implementations may span from eight to 2048 processors, with peak computing power ranging from 400 MFlops to 100 GFlops.

The floating point processors are VLIW pipelined devices optimised to deliver the result of a so called "normal" operation, i.e.

$$A = B * C + D \tag{1}$$

once at each clock time.

The Quadrics system can be programmed in TAO, a FORTRAN-like data-parallel language. This language can be modified and extended through the dynamical ZZ parser on which the compiler is grounded. In this way the user can create a custom meta-language which allows a much more easy and "natural" programming.

Neural Networks and Back Propagation

An artificial neural network is composed by a big array of simple non linear elements (the so called neurons), highly interconnected by links of different weight. These elements decide their own state according to the weighted sum of all the signals incoming from the other neurons. It is evident the similarity to the activity of the neurons in the nervous system.

The y_i state of the i neuron is computed by the:

$$y_i = g\left(\sum_{j=1}^{N} w_{ij} x_j - \theta_i \right) \tag{2}$$

where w_{ij} is the weight of the connection from element i to element j, x_j is the value of activation of neuron j and θ_i is a threshold value for the activation of neuron i.

The function $g()$ is the:

$$g(x) = \frac{1}{1+e^{-x}}. \tag{3}$$

Out of the high number of connections and elements and their non linearity, it is possible to obtain any desired value as output while presenting a given input. In other words, changing the values of the connections it is possible to compute any function of the input. For greater details see e.g. [5] or [7].

The backpropagation is a training algorithm for a huge class of neural networks: the multi layer perceptrons. This architecture is one of the most versatile, allowing the resolution of non linearly separable problems. Through this procedure it is possible to train a network to give the correct association between input and output pairs.

To describe the algorithm let us assume a three layer network, composed of an input layer (i), a hidden layer (j) and an output one (k). We desire the target pattern t_k as output whenever we present as input the pattern p_i.

Through equations 2 and 3 we compute the activations of the hidden layer h_j, then the values for the output one o_k.

It is now possible to compute the error made by the network on pattern p:

$$E_p = \sum_k (t_k - o_k)^2 \tag{4}$$

In order to lower this error, it is necessary to change the connections between the output and hidden layer by the:

$$\delta_k = (t_k - o_k) o_k (1 - o_k) \tag{5}$$
$$\Delta w_{jk} = \eta \delta_k h_j$$

while to modify the connections between input and hidden layer we use the:

$$\delta_j = h_j (1 - h_j) \sum_k (\delta_k w_{kj}) \tag{6}$$
$$\Delta w_{ij} = \eta \delta_j p_i$$

The summation in equation 6 represents the actual back propagation of the error signal from the above layer, from which this algorithm takes its name. Here η is a "learning rate" of the system.

This procedure is repeated for each pattern in the learning set and repeated as well until the total error over all the patterns is small enough. At this point we possess an artificial neural network capable of correctly classify the training patterns

It is possible to show that this algorithm performs a steepest descent optimisation over the total error function:

$$E = \frac{1}{2} \sum_p E_p = \frac{1}{2} \sum_p \sum_k (t_k - o_k)^2 \tag{7}$$

In order to have a faster convergence of the learning algorithm, we have also implemented the so called momentum term in the update of the weights:

$$\Delta w_{ij}^{new} = \Delta w_{ij} + \beta \Delta w_{ij}^{old} \tag{8}$$

this means that the current update will remember the direction of the previous one as scaled by the momentum rate $\beta < 1$.

The Parallel Algorithm

The parallel implementation of a backpropagation learning algorithm could be done in two ways:

1. one can parallelise on the neurons, giving subsections of the network to each processor, keeping the presentation of the training patterns serial,

2. one can create N clones of the network, one per processor, keeping them serial and parallelise on the presentation of the training patterns.

The latter approach have been implemented, see also [6], the main reasons for this being the following. First of all our typical target system is going to be an embedded one of small to medium size (i.e. in the hundreds of neurons) for real time operations on sensory input data (e.g. the vision sub system of a robotised vehicle or a pattern classification system). Hence the memory needs could be granted by such an approach having at our disposition a sufficiently large quantity of memory per processor. Secondly this approach represents the real backpropagation as originally introduced, while the serial presentation of the training patterns is an approximation introduced out of lack of parallel hardware (see [7]).

We are well aware, though, that this approach is memory limited, if one needs to implement a very large network it is possible to run out of local memory, since the number of connections grows with the square of the number of neurons. But for our purposes, as above said, the hardware memory characteristics of Quadrics (1 Mword) is more than sufficient [9].

We have exploited the features of the dynamic parser ZZ in order to create some new statements to automatically manage the declaration of and gain transparent access to the many arrays and variables needed to represent and train the neural network (e.g. connection weights, state values, etc.). In this way it has been possible to perform a linear and "clean" programming of the source code.

Presently, the TAO Neural Network Library is directed towards parallel training of Multi Layer Perceptrons via the Back Propagation learning algorithm.

This library is composed by a set of statements which can be accessed by the end user to create and manage the data structures of the neural network.

In order to find more details on the ZZ implementation of the backpropagation directives see [1].

All the forward and backward operations of the backpropagation are completely local to the Quadrics nodes, allowing parallel classification and training of a network on different input vectors. On the contrary, the update phase performs the global sum of the corrections computed on all the nodes, and applies them to each network replica. This step is the only one requiring communications.

Experimental Measures

In order to experimentally measure the performances of our library, we had at our disposition three different machines of the Quadrics series, a Q1, a Q4 and a QH4, respectively composed of 8 (2 by 2 by 2), 32 (2 by 2 by 8) and 512 (8 by 8 by 8) processors.

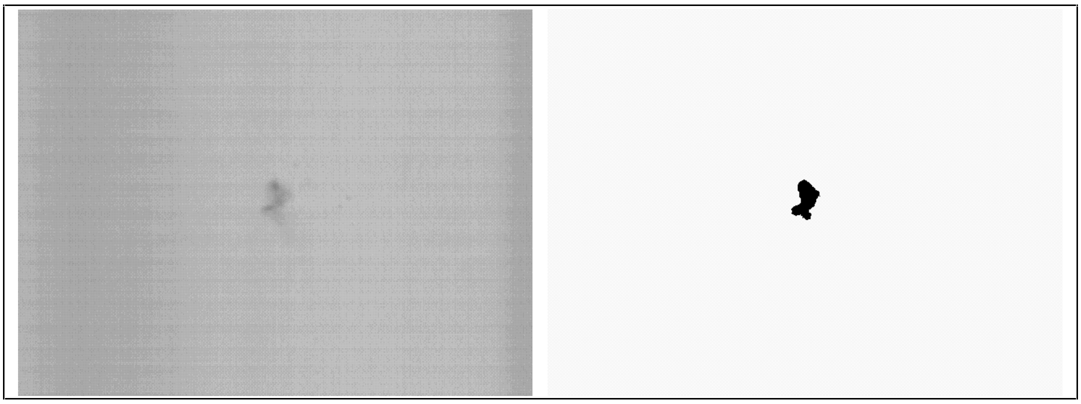
The following timing measures are made on a first test network composed of 20 input neurons, 20 hidden ones and 10 output neurons (a total of 630 connections).

Naturally, the performance reached through our TAO Neural Network Library greatly depends on the network structure, on the machine used and on the number of training examples per processor. In any case we can provide a range of performance for our implementation on the discussed topology.

The tipical measure unit to evaluate neural network learning performance is the *cups*, which stands for connection update per second. The lower end of this range is represented by a figure of *1.94 Mcups*, obtained on a Q1 machine (8 processors) with only one training example per processor on a 20-20-10 network. The upper end on the same network structure is represented by a figure of *499.09 Mcups*, obtained on a QH4 machine (512 processors) with 50 examples per processor.

The HIPERCLASS project

HIPERCLASS is a project of the European ESPRIT Special Action named CAPRI (Calcolo ad Alte Prestazioni per le Imprese) that is aimed at the diffusion of the Parallel Computing among the italian industries. Aim of HIPERCLASS is the porting of the image treatment and defect classification of the surface inspection pilot system designed by CSM onto the parallel hardware and software environment of Quadrics, see figure 1. The technical limit of the present system, is hidden in the conceived architecture. In fact, a set of processing functions has been hardwired in order to get high computing power, but, at the same time, they are practically unmodifiable and that limits the versatility in the application. Besides, the classification stage is performed off line in a subsequent time. Such a porting improves the system flexibility and allows an easier approach to a wider range of applications, since Quadrics represents a more flexible architecture based on a less specialised but still efficient hardware driven by standard software.



X (min,max): 303, 339 Y (min,max): 219, 269 Num.pixel: 942
Xper: 44 Yper: 54 LSum: 97140 Lmin: 72 Lmax: 120

Figure 1. A typical input frame (left) and the Image Processing algorithms output (right). Below some of the features computed by the system ("per" is half perimeter, "L" is luminosity and "LSum" the total luminosity).

The HIPERCLASS inspection system is composed of an image processing module, of an object tracker module and, finally, of a neural network based defect classifier.

The image processing module scans the input data looking for the defectivity of the single pixels on the basis of deviations from average luminosity values, performing also local filtering operations aimed at hole filling and/or elimination of isolated defective pixels.

The object tracker module tracks the defective pixels into objects, taking into account the possible merging on neighbouring defects, and measuring their geometrical and luminosity features.

Finally the neural network based classifier module computes some derived features, with the due normalization, and classifies the defect assigning the class associated to the highest output, otherwise an "unknown defect" signal is output. In order to implement the neural classifier, the above presented SW simulation library has been used.

The data for the neural networks training and test come from the features computed by the object tracker, but the available data features must be pre-processed in order to be given as input to the neural classifier. Depending on the kind of defect, from 8 to 31 of the features are considered as actual input of the given neural network.

The neural classifier architecture is composed of a pool of neural nets (presently nine), each one having the same basic structure of a Multi-Layer Perceptron, trained via the backpropagation algorithm. Each network is dedicated to a single defect class. For certain classes there are more nets, depending on the steel grade the input defect belongs to. All the nets have just one output value. The classification stage is represented by the running of the pool of networks on the feature data following the threshold rule. The correct class of defect will be the one showing the maximum output value crossing a given threshold.

The classifier implementation has been done using an optimized version of the general purpose neural simulation library, where the optimization mainly involved the memory allocation and the internal system variables.

The experimental data file, used for training the classifier, is composed of 1965 patterns. For each net, a training and a test file have been created. A criterion of 2/3 has been employed, that is 2/3 of the available data (randomly selected) for each class has been used for training purposes, the remaining 1/3 has been used for test.

Once the various networks have been trained, the classification has been made by allocating the entire classifier structure on each processor of the Quadrics machine, so that each node is able to classify its defect pattern produced by the image processing phase. As above mentioned, due to a load balancing problem, because the number of defects produced by the image processing module is different on each processor, a re-shuffling process has been implemented, able to distribute the same number of defect patterns to be classified on each processor.

From the classification point of view, a noticeable performance in terms of number of classified objects per second has been reached. A single classification cycle takes approximately 1 ms. Thanks to the defect load balance among processors it is possible to classify about N kilo-objects per second (N=number of processors); this is far enough to Hiperclass purposes, where an estimated defect rate in the order of at least 100 objects per second is given.

Table 1 summarises the results obtained by the neural classifier in terms of recognition performance. Percentages are calculated with reference to total class size.

Defect class	Test set
A	56.96%
B	78.26%
C	74.73%
D	94.74%
E	52.63%
F steel grade 1	50%
F steel grade 2	75%
F steel grade 3	70.83%

Table 1. Recognition performance of the various nets composing the classifier.

These results are the same as during the neural architecture experimentation made in a previous work [10]; they indicate an average defect recognition rate of about 70%.

The classification phase is fully parallel, as long as the number of defects in a given moment is comparable to the number of the available processors. In fact each processor holds a full copy of the classifier and proper code performs the load balance distributing defects among processors. These results demonstrate that the Neural Classification performance is in excess and negligible in relation to the overall performance.

Concerning the computing speed, the overall performance of HIPERCLASS data treatment on a defect free surface is around 34 Mpixel per second. This is due to the skipping of the most part of the processing when no defects are present on the surface. The average performance on a defective frame drops to around 8 ÷ 10 Mpixel/s according with the defect density and distribution in the image. Therefore the data crunching software is able to sustain around 15 Mpixel per second as an average performance, as soon as the defect rate of the rolled steel strip is under one out of two images with a couple of defects each, or, equivalently, under a level of 100 defects per second.

Conclusions

The results show that the discussed implementation of the Multi Layer Perceptron trained by Backpropagation on the Quadrics parallel supercomputers attains an extremely interesting performance. As a consequence, the well known burden of the accurate tuning of parameters and topology of a neural network can be greatly reduced.

This has been evident in the employement of such a SW library in a practical problem, offered by the European Project HIPERCLASS. Through the use of this library, the productivity of the computer scientist is greatly enhanced, allowing a faster testing of network topologies and parameters.

Future work on the library will include its use in different fields of application, e.g. the prediction of pollution concentrations in the lower atmosphere in urban environments, and its further refinement. Another issue to be addressed will be the inclusion of other network architectures in the library, e.g. the Kohonen network.

References

[1] S. Taraglio, F. Massaioli, "An efficient implementation of a backpropagation learning algorithm on a quadrics parallel supercomputer", 1995, *Proceedings of HPCN - Europe 1995, Lecture Notes in Computer Science* **919**, pag 664, Springer-Verlag.

[2] R. Falessi, M.P. Pensini, G. Rovigatti, S. Taraglio, "HIPERCLASS: High Performance Industrial Inspection and Defect Classification in Steel Industry", 1997, *Proceedings of HPCN - Europe 1997*, in press.

[3] A. Bartoloni et al., "A hardware implementation of the APE100 architecture", 1993, *International Journal of Modern Physics C*, **4**, No. 5, 969-976.

[4] A. Bartoloni et al., "The software of the APE100 processor", 1993, *International Journal of Modern Physics C*, **4**, No. 5, 955-967.

[5] J. L. Mc Clelland, D. E. Rumelhart, "Parallel Distributed Processing", 1987, MIT Press: Cambridge, Massachussets, U.S.A..

[6] S. L. Hung, H. Adeli, "Parallel backpropagation learning algorithms on CRAY Y-MP8/864 supercomputer", 1993, *Neurocomputing*, **5**, 287.

[7] S. I. Gallant, "Neural Network Learning and Expert Systems", 1993, MIT Press: Cambridge, Massachussets, U.S.A..

[8] E. J. H. Kerckhoffs, F.W. Wedman, E.E.E. Frietman, "Speeding up backpropagation training on a hypercube computer", 1992, *Neurocomputing*, **4**, 43.

[9] S. Taraglio, "Note sui primi esperimenti di supercalcolo parallelo in ENEA: le Reti Neurali", 1994, ENEA Technical Report, RT/INN/94/41.

[10] R. Falessi, M. P. Pensini, G. Marijuan, “Cold Rolled Flat Products Recognition and Classification through Neural Network Architecture”, 1995, ECSC agreements 7210-GB/410; 7210-GB/411; 7210-GB/932, Final Report