Performance Prediction of large MIMD Systems for Parallel Neural Network Simulations

Louis Vuurpijl, Theo Schouten, Jan Vytopil
Faculty of Mathematics and Informatics, University of Nijmegen, The Netherlands, email: louis@cs.kun.nl

Abstract

In this paper, we present a performance prediction model for indicating the performance range of MIMD parallel processor systems for neural network simulations. The model expresses the total execution time of a simulation as a function of the execution times of a small number of kernel functions, which have to be measured on only one processor and one physical communication link. The functions depend on the type of neural network, its geometry, decomposition and the connection structure of the MIMD machine. Using the model, the execution time, speedup, scalability and efficiency of large MIMD systems can be predicted. The model is validated quantitatively by applying it to two popular neural networks, backpropagation and the Kohonen self-organizing feature map, decomposed on a GCel-512 [1], a 512 transputer system. Measurements are taken from network simulations decomposed via dataset and network decomposition techniques. Agreement of the model with the measurements is within 1%-14%. Estimates are given for the performances that can be expected for the new T9000 transputer systems. The presented method can also be used for other application areas such as image processing.

Keywords. Performance prediction; Parallel neural network simulations; MIMD transputer systems

1. Introduction

When discussing the suitability of an execution platform for a certain application domain, several questions arise. The first is what performance can be achieved for a given machine configuration and kind and size of application. The second is whether by increasing the amount of computing and communication power, the total execution time for a given application can be reduced, i.e. what speedups can be achieved. A third question concerns the scalability of the platform and application, i.e. does the execution time stay constant if the size of both the application and platform is scaled up? In this paper, a method is discussed for predicting the performance of MIMD multi-processor systems for parallel neural network simulations. Using the method, also the expected speedups, scalability and efficiency can be predicted.

One approach to predict the performance of an execution platform for a particular application is to compare it with applications or benchmarks for which the performance is already measured on (part of) the platform. The performance of the benchmark that best matches the application could then be used as an indication. Unfortunately, comparing neural networks with existing applications is not feasible, as there exists a wide variety of different neural network algorithms and an even wider variety in their implementations. Comparing for example the performance of one system expressed in MCUPS (million connection updates per second) with another, is meaningless unless parameters like the complexity of the neural network algorithm, the programming language, compiler, library routines, implementation technique and amount of optimization efforts are precisely mentioned. This involves that if this approach is taken, the selected benchmarks have to be examined in great detail, which in general puts too much efforts on an application programmer. Furthermore, exist-

[1] We would like to acknowledge the University of Amsterdam and Parsytec Gmbh for allowing us to use the GCel located in Amsterdam during the CAMPP'93 programme.

ing benchmarking methods are targeted on vector supercomputers and not on MIMD parallel processor architectures, although the Genesis [6] benchmarks have implemented a range of kernel and small application benchmarks on MIMD architectures. Finally, as none of these match the characteristics of distributed neural network simulations, some other method is required that is targeted on neural networks and machines with MIMD architectures.

In our approach, the execution time of a neural network simulation with n neurons, w connections and p patterns running on a platform with P processors is modeled as the sum of calculation and communication times:

$$T(P,n,w,p) = T_{calc}(P,n,w,p) + T_{comm}(P,n,w,p) \quad (1)$$

Overlap of communication and calculation is usually not feasible on MIMD architectures, as in our experience this leads to very complex programs and the decrease in execution time compared to (1) is small. The calculation time T_{calc} can be modeled as consisting of the times required for executing a small number of *kernel* functions (e.g. compute a weight change, update a connection, update a neuron). These are executed a large number of times (e.g. per connection or per neuron):

$$T_{calc}(P,n,w,p) = \frac{1}{P} \sum_i N_i(n,w,p)\; t_i \quad (2)$$

where t_i is the execution time for computing function i and $N_i(n, w, p)$ denotes the number of times the function is computed. Note that a perfect load balance can be assumed, as either n,w or p are large compared to P. The communication time is modeled as the number of times a single information unit (e.g. a connection or activation value) must be sent over a physical communication link $(C(P, n, w, p))$, multiplied by the time required for communicating one value.

$$T_{comm}(P,n,w,p) = C(P,n,w,p)\; t_{comm} \quad (3)$$

The setup time of a communication can be neglected as usually a large number of bytes are sent at once. If required by the hardware resources or communication patterns of the neural network, a more complicated model can be used.

The needed benchmarks in our method are restricted to measuring the execution time of a small number of kernel functions on one processor and the time needed to communicate a single information unit between two processors. This approach can be classified as kernel benchmarking [2, 6].

The problem of decomposing a given neural network over a parallel processor system with given topology has often been addressed in the literature. Chu and Wah [4], Witbrock and Zagha [15] and various other authors have discussed the implementation of backpropagation [11] networks on diverse parallel processor systems. Similar efforts have been reported implementing Kohonen [8] networks by for example Obermayer *et al* [10] and Wu [16]. Several other neural networks like Hopfield [7] networks [**?**, 5] have also been implemented on parallel architectures.

Methods for decomposing neural networks on MIMD multi-processor systems can be distinguished in three levels of decomposition, *job-level* decomposition, *dataset* decomposition and what we call *network* decomposition. The first two techniques are coarse grained decomposition methods which are well suited for MIMD parallel computer architectures, as the job sizes are large and the amount of inter-processor communications is small. The last technique is defined as the collection of decomposition methods that can be used to divide a neural network onto a processor system. In general, these techniques are more fine grained than the former ones and thus require more inter-processor communications. In the subsequent sections, our method is validated quantitatively for full applications by discussing two parallel neural network algorithms decomposed on a GCel-512, a multi-transputer system having 512 processors arranged in a 16x32 grid configuration. In particular, the performance of the GCel for backpropagation and Kohonen neural networks using dataset and network decomposition is presented.

2. Dataset decomposition

Dataset decomposition is a decomposition technique that places complete copies of a neural network on each processor. Each network copy is initiated with the same parameters and architecture, so the same neural network is present on every processor. The parallelism that is exploited in this technique stems from the dataset. The dataset is equally divided over the available processors and each neural network copy operates on its local subset. Especially if the datasets are large, dataset decomposition is a highly efficient parallelization technique as during the computation of the data subsets no communication is necessary.

2.1. *Calculation times*

Algorithm 1 describes the heuristics for dataset decomposed parallel neural networks (PNNs). First, the dataset is distributed over the available processors (step 1). Then, a number of *epochs* are performed. During each epoch, each network copy is trained with all local patterns and computes its weight changes based on these (step 2 is performed for each pattern). After learning all local patterns, the weight changes are communicated (step 3). Finally, in step 4 each network copy updates its connections based on the accumulated weight changes. Steps 2 to 4 are repeated until the number of epochs has passed a certain value or the neural network has reached a certain error criterion.

```
step 1:  distribute_patterns();
         while (not_ready) {
           for (all patterns p)
step 2:      compute_weight_changes(p);
step 3:    communicate_weight_changes();
step 4:    update_connections();
         }
```

Algorithm 1: Dataset decomposition.

The total calculation time per epoch consist of the time for step 2 times the number of patterns and the time for step 4 as given in (4).

$$T_{calc}(p,n,w) \quad = \quad p\,T_2(n,w) + T_4(n,w) \qquad (4)$$

Benchmarking these times requires the identification of the basic neural network operations that are performed in steps 2 and 4. The total calculation time can then be expressed in terms of the number of patterns and the neural network size, i.e. the number of connections and/or the number of neurons in the network. For any machine configuration and compiler used, this can be done.

We have measured the typical routines for backpropagation and Kohonen neural networks on one GCel-512 node. The GCel-512 is a 512 multi-transputer machine arranged in a 16x32 grid topology. It is controlled by the Parix operating system. Each node is a T805 transputer running on 30MHz with 4MByte RAM. All programs used for the subsequent measurements run on this machine and are compiled with the ACE expert C compiler without any optimization options. The software that was written has no specific optimization techniques such as efficient use of stack or on-chip memory. The measured calculation times for backpropagation networks have a linear dependency on the number of connections in the neural network, as expressed below. For the Kohonen neural network, the calculation times in step 4 also depend on the number of neurons. For a further explanation of the typical operations that make up the calculations, see [14]. All subsequent times are expressed in μseconds.

$$\begin{aligned} T_{Back}(p,n,w) &= p\ w\ 16.36 + w\ 11.56 \\ T_{Koho}(p,n,w) &= p\ (w\ 8.69 + n\ 20.7) + w\ 5.33 \end{aligned}$$

Derived from these calculation times, the performance of a single GCel transputer for dataset decomposed backpropagation and Kohonen networks can be estimated by dividing the number of connections that are computed (CU, connection updates) by the calculation time. As the problem size depends on the number of patterns, the maximum performance on a single T805 transputer for the backpropagation network can be expressed as in (5).

$$\lim_{p\to\infty} \frac{p\ w}{T_{Back}(p,n,w)} = \frac{1}{16.36}\ CU/\mu sec \approx 0.061 \text{ MCUPS} \quad (5)$$

Similarly, the performance of a single GCel transputer for dataset decomposed Kohonen networks can be estimated as $1/8.69\ \ CU/\mu sec \approx$ 0.115 MCUPS, using the fact that n is small compared to w. As for dataset decomposition near linear speedups can be achieved, the maximal expected performance of backpropagation and Kohonen on the GCel-512 are 31.23 and 58.88 MCUPS respectively.

2.2. *Communication times*

The neural network implementations used in this paper use a so called master-slave decomposition. Each processor runs a slave process that manages part of the simulation. The master process runs on processor (0,0) in the grid, and controls the slave processes while managing its own part of the simulation. In [13] and [14], a detailed description of the different patterns of communication required for such parallel neural network simulations is given. It is explained that the number of communications can be reduced enormously by using only local communications. This can be established via broadcast and gather operations. During the synchronization phase (step 3 in algorithm 1), all weight changes computed on each processor have to be added up and made available to all other processors. This is performed by subsequently gathering and accumulating the weight changes from every slave processor to the master processor and then broadcasting the accumulated information from the master to the slaves. During the gathering also the accumulation of the weight changes can take place, which can be done in parallel.
For these kind of gather-accumulate and broadcast operations, a tree topology is optimal [13]. Unfortunately, the GCel does not support the physical configuration of tree topologies. We have tried to use virtual tree topologies by using the Parix `MakeTree` utility, but the performance of the communication was far worse than using the physical grid topology and communicating locally. Furthermore, the mapping of a tree on the GCel grid gave unpredictable results. One would expect that the larger the grid on which a tree can be mapped, the better the mapping result. However, this appeared not to be the case. Each broadcast operation from the master to every slave requires $W + H - 2$ communications. Each gather and accumulation operation requires the same amount. Note that no packeting of messages is used in this model. Each message is received entirely before it is sent to neighbouring processors. Though dividing the message in packets and receiving and sending it in parts is faster, this requires more programming efforts and involves that the communication model becomes more complicated. However, the number given above can always be used as an upper limit.

The communication benchmarks required for our model only consist of measurements over one physical communication link. The raw communication time for communicating w connections is expressed as $w\ t_{comm}$. The required accumulation benchmarks are performed on one processor and consist of the time needed to sum up two amounts of w connections, which is expressed as $w\ t_{acc}$. The total time for broadcasting (B), gathering (G) and accumulating (A) is given by $GAB(W, H, w)$. The equations below are a more precise representation of (3).

$$\begin{aligned} B(W,H,w) &= w\ (W+H-2)\ t_{comm} \\ GA(W,H,w) &= w\ (W+H-2)\ (t_{comm}+t_{acc}) \\ GAB(W,H,w) &= w\ (W+H-2)\ (2\ t_{comm}+t_{acc}) \end{aligned}$$

Similarly, this time can be modeled for ternary tree topologies [14] as:

$$GAB(P,w) = w\ 3\ depth(P)\ (2\ t_{comm}+t_{acc})$$

We have used the `SendLink` and `RecvLink` communication library routines of Parix. These resulted in a communication time of t_{comm} = 3.59 μseconds, where each single information unit is represented with single precision floating point (32 bits). The setup time can usually be neglected, as w is relatively large. For the accumulation time, we have measured $t_{acc} = 5.59$ μseconds. The results of experiments on different grid sizes indicate that GAB predicts the gathering, accumulation and broadcasting within a precision of 4% for 16 processors to 9% for 512 processors.

2.3. *Performance for dataset decomposition*

The performance expressed in MCUPS can be calculated by dividing the number of connections that are updated by the total time:

$$\begin{aligned} T(P,p,n,w) &= \frac{T_{calc}(p,n,w)}{P} + GAB(P,w) \\ MCUPS(P,p,n,w) &= \frac{p\ w}{T(P,p,n,w)} \end{aligned} \quad (6)$$

For each of the experiments reported below, a network size of 100000 connections is used. The backpropagation architecture is a 250x200x250 network, the Kohonen map is 50x50 and has 40 inputs. Figures 1 and 2 depict the measured performance expressed in MCUPS for backpropagation and Kohonen networks. For the former network the measured performance is predicted correctly within 4%-9% (which corresponds to the deviations of the predicted values for the communication and calculation costs). We have measured that even for relatively small (1 Kweights) networks, the prediction appears to be correct within 9%. For Kohonen networks, the performance can be predicted with even higher precision (correct within 1%), as the calculation times are modeled more accurate. As predicted in section 2.1, the performance of Kohonen networks is about twice the performance of backpropagation networks, merely because the latter needs more computational efforts than the former.

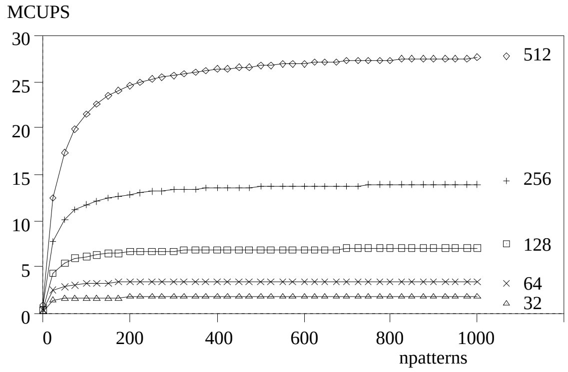

Figure 1: Performance for backprop networks.

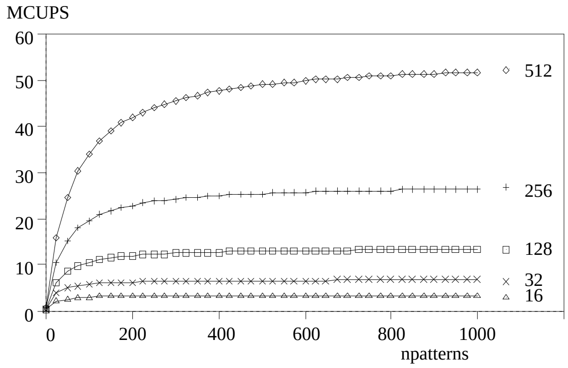

Figure 2: Performance for Kohonen networks.

2.4. *Speedup, efficiency and scalability*

Using the predicted execution times, the performance prediction model can be used to estimate the expected speedup, scalability and efficiency. The speedup for a problem with n neurons and w connections running on $P = W\ H$ processors is defined as the time it takes to run the problem on one processor divided by the time it takes to run the problem on P processors:

$$S(P,p,n,w) = T(1,p,n,w)/T(P,p,n,w) \quad (7)$$

For Kohonen and backpropagation parallel neural network simulations, the measured speedups are depicted below for different problem sizes (1-1000 patterns per processor). Again, the predictions are within a deviation of 9%. Note that the larger the problem size, the higher the speedups that are achieved, which is due to the communication computation trade-off. The more work each processor has to do, the less the performance is influenced by the communication overheads. For small sized problems (i.e. less than 10 patterns per processor), the speedup limit is reached.

Now consider again how the speedup is defined, assuming that the processor system consists of a $\sqrt{P} \times \sqrt{P}$ grid. From the definition

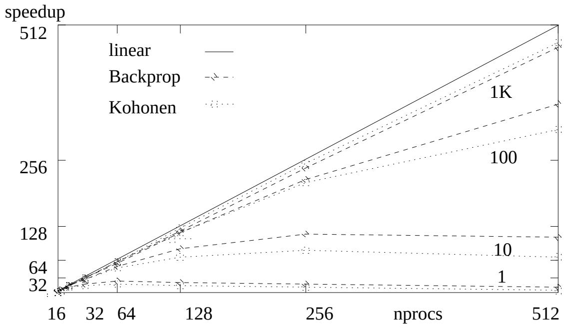

Figure 3: Speedups dataset decomposition.

of $GAB(P, w)$, and with $T_o(w)$ representing the overhead for accumulating and communicating w connections, the speedup $S(P, p, n, w)$ is:

$$\frac{T(1, p, n, w)}{T(1, p, n, w)/P + 2\ T_o(w)\ (\sqrt{P} - 1)}$$

By taking the first derivative of $S(P)$ and solving $S'(P) = 0$, we obtain the optimum number of processors (for which we achieve the highest speedup):

$$P^o \quad = \quad \sqrt[3]{\frac{T(1, p, n, w)^2}{T_o(w)^2}}$$

For backpropagation and Kohonen networks with 1 and 10 patterns per processor, the optimum number of processors following this formula is 75 and 300 and 52 and 239 respectively. These correspond to the measured speedups as depicted in figure 3.

The efficiency of a parallel program is defined as $E(P, p, n, w) = S(P, p, n, w)/P$. Ideally, the efficiency equals 1 which corresponds to having linear speedup. As can be observed in figure 3, parallel neural network simulations decomposed via dataset decomposition can achieve high efficiencies depending on the load per processor. For example a backpropagation network running on 512 processors has efficiencies ranging from $E(512, 512, n, w) \approx 0.03$ up to $E(512, 512K, n, w) \approx 0.92$, where (n, w) corresponds to the neural networks as specified above.

The speedup as discussed above is also known as fixed-size speedup. The problem size is kept constant, and the number of processors is increased. If, as often is the case in simulation studies, researchers want to increase their problem size (e.g. grid size in weather forecasting or image processing, or the number of patterns or connections in artificial neural networks), they want to know if their application is scalable on a specific machine architecture. When examining the scalability of a parallel program, the problem size is not kept fixed but instead, it is increased linearly with the amount of processors. A parallel program is scalable if when increasing both the problem size and the number of processors, the total execution time of the parallel program stays constant. This holds if what we call the scalability factor equals one:

$$fs(P, k, p, n, w) = \frac{T(P, p, n, w)}{T(k\ P, k\ p, n, w)} \tag{8}$$

For the networks considered above, scalability factors of for example $fs(4, 128, 25, n, w) = .75$, $fs(4, 128, 250, n, w) = .97$ and for scaling from 256 to 512 processors up to $fs(256, 2, 4, n, w) = .95$ have been predicted correctly within a deviation of 9%. Note that these factors indicate that dataset decomposed neural networks are indeed highly scalable.

3. Network decomposition

If a neural network model does not accommodate epoch learning or if it does not fit on one processor, dataset decomposition cannot be exploited. The first case holds for networks that require updates of the weights immediately after being inputted with a training sample, such as Hopfield [7] or the ART [3] networks. The second case is obvious: if a neural network requires more memory for its connections than there is available on one processor, more processors have to be used. In these cases, the neural network has to be decomposed via network decomposition. Like with dataset decomposition, the implementations of network decomposed PNNs that are discussed in this section consist of a master process and several slave processes. For MIMD parallel computers like a transputer system, this involves that if processes require neural network information managed by other processes, they have to exchange the required information via communication. A general parallel algorithm for network decomposed PNNs is given below. In step 1, a new training sample is broadcasted to all slave processors. For many neural network models, step 2 can be computed without further communication. For example Kohonen and ART networks compute the new network state based only on the input patterns. For backpropagation and Hopfield networks, the computation is done in a number of iterations. During each iteration, the new state of (a number of) the neurons in the neural network is computed. As the neurons are divided over the processors and each neuron needs information of the neurons that have a connection to it, after each iteration the (partly) computed network state has to be communicated between the processors. This involves that for these networks, steps 2 and 3 must be repeated a number of times. Based on the new network state, the weight changes are computed in step 4. For backpropagation networks this is done by repeating steps 4 and 5 for each layer in the network. After making the weight change information available globally (step 5), the actual changing of the weights is performed (step 6).

```
while (not_ready) {
  step 1:  clamp_input();
  step 2:  compute_new_netstatus();
  step 3:  communicate_netstatus();
  step 4:  compute_weight_changes();
  step 5:  commmunicate_change_info();
  step 6:  change_weights();
}
```

Algorithm 2: Network decomposition.

3.1. *Decomposing the backpropagation network*

The backpropagation neural network has a layered architecture with one input layer, a number of hidden layers and an output layer. Two subsequent layers in the network are fully connected and the network operates in two passes. During the forward pass, for each layer l the activations of the neurons in l (denoted by n_l) are computed based on the activations of the previous layer. During the backward pass, for the output layer and each hidden layer, the errors that were made are computed and propagated back, hence the name backpropagation. Using geometric decomposition, each layer is equally distributed over the available slave processors, thus ensuring that the load is well balanced. For computing the activation of a neuron residing on a slave processor, the activations of the neuron's previous layer are required. This means that an all-to-all broadcast of all activations in a layer is needed. Again, this is implemented by subsequently gathering and broadcasting messages. In order to reduce the amount of communications needed during the gathering phase, messages are combined locally at each node as depicted in figure 4.

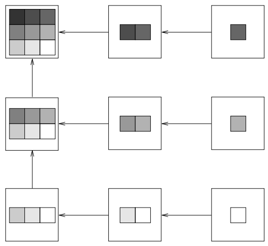

Figure 4: Gathering activations on a grid.

As can be observed, the number of activations per layer that are gathered from right to left in a $W \times H$ grid equals N_W (see 9), where it is assumed that each processor hosts $n_l/(W\ H)$ neurons. In the leftmost column of the grid, the activations are gathered upwards, which amounts to N_H (10), so in total the number of activations that are communicated is $N(W, H, n_l)$ (11). The time for gathering and broadcasting the n_l activations in layer l is given by T_l and the total time for all layers is given by (12):

$$
\begin{aligned}
N_W(W, H, n_l) &= \frac{n_l}{W\ H} \sum_{i=1}^{W-1} i \\
&= \frac{n_l}{H} \frac{W-1}{2}
\end{aligned}
\tag{9}
$$

$$
\begin{aligned}
N_H(W,H,n_l) &= \frac{n_l}{W\ H}\sum_{i=1}^{H-1}(i\ W) \\
&= \frac{n_l}{H}\frac{H\ (H-1)}{2} && (10) \\
N(W,H,n_l) &= \frac{n_l\ (W-1+H\ (H-1))}{2\ H} && (11) \\
T_l(W,H,n_l) &= t_{comm}\ N(W,H,n_l) + B(W,H,n_l) \\
T_L(W,H,L) &= \sum_{l=1}^{L-1} T_l(W,H,n_l) \ = \\
t_{comm}\ (n-n_0)\ (\ W &+ \frac{W-1}{2\ H} + \frac{3}{2}\ (H-1)\ ) && (12)
\end{aligned}
$$

Synchronization step 5 in algorithm 2 can be performed by using the gather, accumulate and broadcast communication operations as defined in section 2. For all layers this involves communicating $n-n_0$ values. The total communication time $T_{commback}(W,H,L,n,w)$ can be estimated using (12) and $GAB(W,H,n-n_0)$:

$$
t_{comm}\ (n-n_0)\ (\ W + \frac{W-1}{2\ H} + \frac{3}{2}\ (H-1)\ + \\
(W+H-2)\ (2 + t_{acc}/t_{comm})\ )
$$

A similar estimation is given in [14] for the communication time on ternary tree topologies with depth D:

$$
t_{comm}\ (n-n_0)(3\ (D-1)\ (1+t_{acc}/t_{comm}) \\
+\tfrac{1}{4}\ (4\ D-1)\ 3^D - 6\ D - 3)
$$

In order to determine the calculation time, the times for steps 2, 4 and 6 were analyzed and measured depending on the number of neurons in two subsequent layers. The following relations are found (see also [14]). In step 2, for each two layers l and $l+1$, the time for computing the new net status amounts to 5.7 $(n_l\ n_{l+1})$ μseconds. For all layers, this means 5.7 w μseconds. In step 4, two main calculations are made. The first is the computation of the errors in a layer made during a forward pass, which depend on the number of neurons in the layer: 24.2 n_l μseconds. The second is the computation of the weight changes based on the computed errors, which depends on the number of weights in the network: 2.56 w μseconds. Adjusting the weights in step 6 costs 9.8 w μseconds. The total time on a $W \times H$ grid can be expressed as:

$$
T_{commback}(W,H,L,n,w) + \frac{18.06\ w + 24.2\ (n-n_0)}{W\ H} \quad (13)
$$

3.2. *Performance for backpropagation*

The performance model described above is validated quantitatively via a number of experiments carried out on the GCel-512. The training times for large backpropagation networks (sizes 375K-200M) are measured on varying grids. In figure 5, the measured performance for the different neural networks on different grids is depicted. In figure 6, it is shown that the predicted and measured performances for executing the forward and backward pass for backpropagation networks match within 2%-14%) for all experiments.

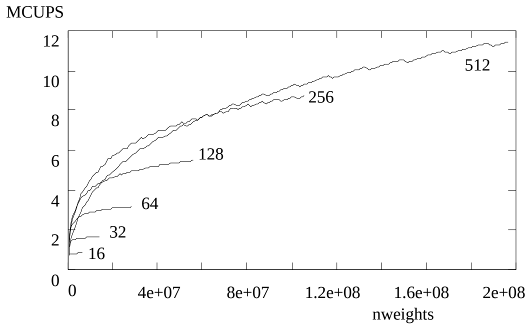

Figure 5: Measured performance.

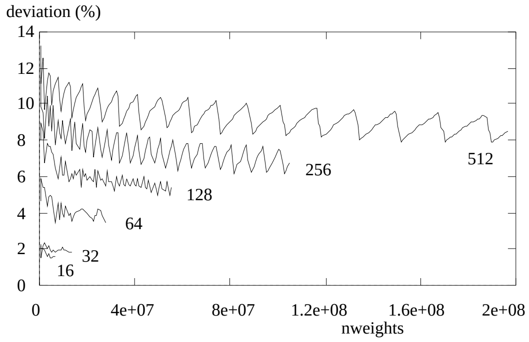

Figure 6: Deviations.

Note that the depicted plots for the deviations in figure 6 have a peculiar characteristic. This is due to the load balance of the processors in the GCel grid. For some neural network architectures, the load balance is not perfect, which means that when increasing the size of the neural network, the time may not increase. The depicted effect can be explained as follows. As the network size increases, the load becomes better balanced, resulting in a relatively smaller execution time. This involves an increase in the deviation. At a certain moment, the network size becomes such that the load suddenly becomes less well balanced, resulting in an increase in execution time and thus a drop in the deviation (as the predicted times are higher than the measured times). This effect leads to the serrate curves.

3.3. *Speedup, scalability and efficiency.*

The expected speedup, scalability and efficiency can be predicted for network decomposed backpropagation networks using (13) within a deviation of 2%-14%. As there is a need for communication for each pattern, the speedup limit is reached much sooner than with dataset decomposition. This can be observed in figure 7, which depicts the measured speedups for backpropagation networks of sizes 375K up to 4.5M connections. For large networks (> 50 Mweights), the speedup limit is not reached, but relatively low efficiencies of about 0.5 are achieved. This involves that computing a large network (50M-100M) on 256 processors takes about the same time as computing it on 512 processors, which could already be observed in figure 5.

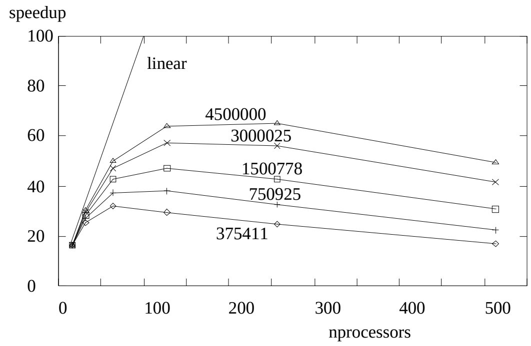

Figure 7: Speedups for backpropagation.

The scalability of network decomposed backpropagation networks is depicted in figure 8. Each line nxw in the figure corresponds to the scalability that is reached when computing the network of k w weights on k n processors. Note that although the scalability is not linear, the scalability limit is not reached. However, from this figure it can be concluded that backpropagation networks scale well only for large neural networks.

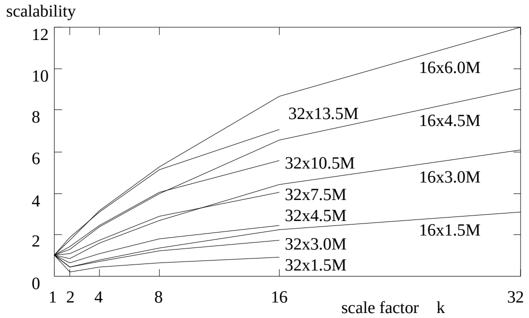

Figure 8: Scalability for backpropagation.

3.4. *Decomposing the Kohonen network*

The Kohonen self-organizing feature map consists of a two-dimensional grid of neurons (the map), with each neuron fully connected to an input layer. Each neuron therefore has as many incoming connections as the number of input features per pattern. Furthermore, neurons have lateral connections with other neurons laying in the same neighbourhood, i.e. within a certain Euclidean distance in the map. For each input pattern $\vec{x}$, all neurons compute the match between their weights $\vec{w_i}$ and $\vec{x}$ (step 2 in algorithm 2). By dividing the neurons over the available processors, step 2 can be computed in parallel without communication. After computing the match for every neuron, the overall best match must be computed in order to find the "winning" neuron c (step 3). For all neurons laying in the same neighbourhood as c, their weights are changed (step 6). Similar to backpropagation networks, the neurons are distributed over the available processors and each neuron holds it incoming connections locally. As all processors have about the same number of neurons, the load for step 2 is well balanced. However, the load in step 6 is not well balanced if we apply geometric decomposition to the Kohonen map. As only the neurons which lay in the neighbourhood of a winning neuron have to change their weights, only the processors hosting these neurons have to perform step 6. Therefore, we use another decomposition technique called scattered decomposition [12], as has been suggested in [16]. Using this technique, neurons laying close to each other in the map are placed on distinct processors, thus guaranteeing that step 6 is also well balanced.

The only communications necessary for the Kohonen map are contained in steps 1 and 3. In step 1, the input pattern $\vec{x}$ is broadcasted to all processors. In step 2, the global winner is gathered and broadcasted. Finding a global maximum or minimum value in a MIMD parallel processor can best be done in a tree topology. Each node in the tree compares its value with the values of its sons and sends the best value to its father. On the GCel grid, this is implemented similarly to the gathering of activations as explained in figure 4. The total communication time for Kohonen networks with an input pattern size s can thus be expressed as $B(W, H, s) + GB(W, H, 1)$. Compared to the calculation time, the required communications for network decomposed Kohonen networks are very low. Therefore, near linear speedups and high scalabilities and efficiencies can be expected. Indeed, when considering figure 9, it can be deduced that this holds. The measured performances agree to the expected values within a precision of 5% for 16 processors to 12% for 512 processors.

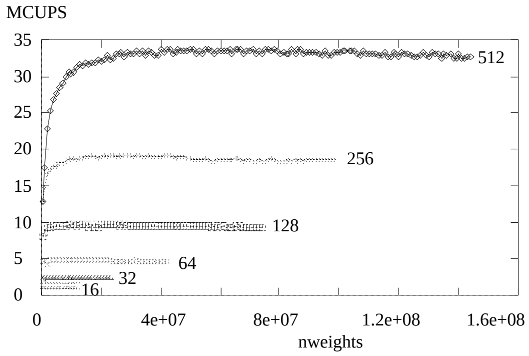

Figure 9: Performance for Kohonen.

4. Conclusions.

We have presented a method for predicting the performance of a MIMD platform for parallel neural network simulations. The model expresses the execution time of a simulation as a function of execution times of a small number of kernel functions. These times need to be measured on only one processor and over one physical communication link. For backpropagation and Kohonen network implementations using both network and dataset decomposition, model functions were fully developed for MIMD grid and tree topologies. From the execution time the performance, speedup, scalability and efficiency for larger processor systems can be derived. We were able to quantitatively validate the model for large processor grids on a GCel-512. The model appeared to be correct within 1%-9% for dataset decomposition and within 2%-14% for network decomposition.

Our model is general purpose because it can be applied to any application for which the overall time can be estimated by a small number of calculation and communication benchmarks. For example many other neural network or parallel image processing applications show this behaviour. The model can also be used to predict the performance of MIMD systems which are not yet available for benchmarking the kernel functions or which are under design. For this the times of kernel functions must be estimated (e.g. from general purpose benchmarks or instruction timings). Furthermore, if other communication networks than a grid are used, also the communication model for broadcast and gather primitives has to be adapted, like we have done for a tree topology.

It is interesting to make some predictions for the T9000 transputer, extrapolating the measured parameters on the T805 by dividing them by the increased computing and communication powers of the T9000. Based on the information in [9] and [1], one could roughly say that the T9000 is about 8 times faster than the T805 and has about 6 times higher bandwidth. This means that the number of processors after which the speedup decreases for dataset decomposition is reduced by a factor $\sqrt[3]{(36/64)} \approx 0.83$. This is due to the fact that the communication time is less reduced than the calculation time. However, when exploiting the new C104 communication chips, it will be cheaper to use all-to-all communications, thus reducing the communication time. From equations (6) and (13), it can be derived that the performance will then increase by a factor 6 to 8, up to 200 and 400 MCUPS for backpropagation and Kohonen networks decomposed via dataset decomposition and 100 to 300 MCUPS for network decomposition. This involves that future transputer systems based on the T9000 and the C104, are even better suited for parallel neural network simulations than current ones.

References

[1] G. Bader and B. Przywara. T9000 - A Preliminary Evaluation of Arithmetic Performance. Brief Note, Interdisziplinares Zentrum fur Wissenschaftliches Rechnen, Universitat Heidelberg, Im Neuenheimer Feld 368, D-6900 Heidelberg, iwr1.iwr.Uni-Heidelberg.de, May 1993.

[2] M. Berry, G. Cybenko, and J. Larson. Scientific Benchmark Characterizations. *Parallel Computing*, 17(2):1173–1194, 1991.

[3] G.A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[4] L-C. Chu and B.W. Wah. Optimal Mapping of Neural Network Learning on Message-Passing Multicomputers. *Journal of Parallel and Distributed Computing*, 14:319–339, 1992.

[5] B.M. Forrest, D. Roweth, N. Stroud, D.J. Wallace, and G.V. Wilson. Implementing Neural Network Models on Parallel Computers. *The Computer Journal*, 30(5):413–419, 1987.

[6] A.J.G. Hey. The Genesis Distributed Memory Benchmarks. *Parallel Computing*, 17(2):1275–1283, 1991.

[7] J.J. Hopfield. Networks and Physical Systems with Emergent Collective Computational Abilities. In *Proc. Natl. Acad. Sci. USA 79*, pages 2554–2558, 1982.

[8] T. Kohonen. *Self-Organization and Associative Memory*. Springer Verlag, Berlin, second edition, 1988.

[9] Inmos/SGS-Thomson Microelectronics. The T9000 Transputer Product Overview, 1991.

[10] K. Obermayer, H.Heller, H. Ritter, and K. Schulten. Simulation of Self-Organizing Neural Nets: a Comparison between a Transputer Ring and a Connection Machine CM-2. In *Proceedings of the Third Conference of NATUG*, Sunnyvale, CA, 1990.

[11] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.

[12] T. Tollenaere and G.A. Orban. Simulating Modular Neural Networks on Message-Passing Multiprocessors. *Parallel Computing*, 17(1):361–379, 1991.

[13] L.G. Vuurpijl and Th.E. Schouten. Suitability of Transputers for Neural Network Simulations. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practice*, pages 528–537. IOS Press, 1992.

[14] L.G. Vuurpijl and Th.E. Schouten. Performance of MIMD Execution Platforms for PNNs: How many MCUPS? Technical report, Department of Real-Time Systems, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, August 1993. In progress.

[15] M. Witbrock and M. Zagha. An Implementation of Backpropagation Learning on GF11, a Large SIMD Parallel Computer. *Parallel Computing*, 14:329–346, 1990.

[16] C-H. Wu, R.E. Hodges, and C-J. Wang. Parallelizing the Self-Organizing Feature Map on Multiprocessor Systems. *Parallel Computing*, 17(1):821–832, 1991.