A Scalable Performance Prediction Method for Parallel Neural Network Simulations

Louis Vuurpijl, Theo Schouten and Jan Vytopil

University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
phone: +31 80 652710, fax: +31 80 553450, email: louis@cs.kun.nl

Abstract. A performance prediction method is presented for indicating the performance range of MIMD parallel processor systems for neural network simulations. The total execution time of a parallel application is modeled as the sum of its calculation and communication times. The method is scalable because based on the times measured on one processor and one communication link, the performance, speedup, and efficiency can be predicted for a larger processor system. It is validated quantitatively by applying it to two popular neural networks, backpropagation and the Kohonen self-organizing feature map, decomposed on a GCel-512, a 512 transputer system. Agreement of the model with the measurements is within 9%.

1 Introduction

In this paper a method is presented for predicting the performance of large MIMD systems for parallel neural network simulations. Using the method, also predictions can be given for the speedups and efficiencies that can be achieved. The total execution time of an application is modeled as the sum of its calculation and communication times. For a neural network simulation with n neurons, w connections and p patterns running on a platform with P processors this is defined as:

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

The calculation time T_{calc} can be modeled as the sum 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 N_i (e.g. per connection or per neuron), where each N_i costs time t_i:

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

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 (modeled as $C(P,n,w,p)$), multiplied by the time required for communicating one value:

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

If required by the hardware resources or communication requirements 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]. In the subsequent sections, our method is validated quantitatively by predicting the performance of the GCel-512 (a 512 multi-transputer system) for backpropagation [6] and Kohonen [3] neural networks decomposed via a technique called dataset decomposition.

2 Dataset decomposition of neural network simulations

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 [1], Witbrock and Zagha [8] and various other authors have discussed the parallel implementation of backpropagation networks. Similar efforts have been reported implementing Kohonen networks by for example Obermayer *et al* [5]. Various techniques can be used to decompose a neural network over a number of processors. In this paper, the attention is focussed on dataset decomposition of backpropagation [6] and Kohonen [3] neural networks. Using this technique, each processor has a copy of the network, initiated with the same parameters and architecture. The parallelism that is exploited in this technique stems from the dataset. The patterns contained in this set are equally divided over the available processors and each network copy operates on its local subset. Especially for large datasets, dataset decomposition is a highly efficient parallelization technique as during the computation of the data subsets no communication is necessary.

2.1 Calculation costs of dataset decomposed neural networks

Algorithm 1 describes the heuristics for dataset decomposed neural networks. 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 them (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.

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

Algorithm 1: General heuristic for dataset decomposed PNNs.

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. The total calculation times per epoch consist of the times for step 2 times the number of patterns and the times for step 4 (4). The equation has as parameters the number of patterns p and the network size expressed in number of neurons n and number of connections w.

$$T_{calc}(p, n, w) = p \cdot T_{step2}(n, w) + T_{step4}(n, w) \tag{4}$$

We have measured the typical routines contained in steps 2 and 4 for backpropagation and Kohonen neural networks on one GCel-512 node (respectively 4 and 3 function kernels). 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 register variables.

2.2 Communication costs for dataset decomposed neural networks

Parallel neural network simulations decomposed via dataset decomposition require all-to-all communications. In step 3 of algorithm 1, every processor needs the weight changes that are computed on every other processor. In [7], a detailed description of the different patterns of communication required for 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. All-to-all communications are then implemented by subsequently gathering and broadcasting messages. In the case of dataset decomposition, 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 [7]. 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.

The neural network implementations used in this paper use a so called master-slave decomposition. The master process runs on processor (0,0) in the grid, the slave processes run on the other processors. The communications required in steps 1 and 3 in algorithm 1 are performed using broadcast and gather primitives. Each broadcast operation from the master to every slave requires

$width + height - 2$ communications. Each gather and accumulation operation requires the same amount. The communication benchmarks required for our model only consist of measurements over one physical communication link. The raw communication times for communicating w connections are expressed as $T_{comm}(w)$. 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 $T_{acc}(w)$. The total times for broadcasting (B), gathering (G) and accumulating (A) are depicted in (5).

$$GAB(P, w) = (width + height - 2) \cdot (2 \cdot T_{comm}(w) + T_{acc}(w))) \tag{5}$$

We have used the `SendLink` and `RecvLink` communication library routines of Parix. These resulted in a communication time of $T_{comm}(w) = 3.59 \cdot w + 50$ μseconds, where each connection is represented with single precision floating point. The setup time can usually be neglected, as w is relatively large. For the accumulation time, we have measured $T_{acc}(w) = 5.59 \cdot w$ μseconds. The results of experiments on different grid sizes indicate that by measuring the communication times for one transputer link and the accumulation times for one transputer enables the prediction of the times on any transputer grid, within a precision of 4%-9%.

2.3 Performance, speedup and efficiency

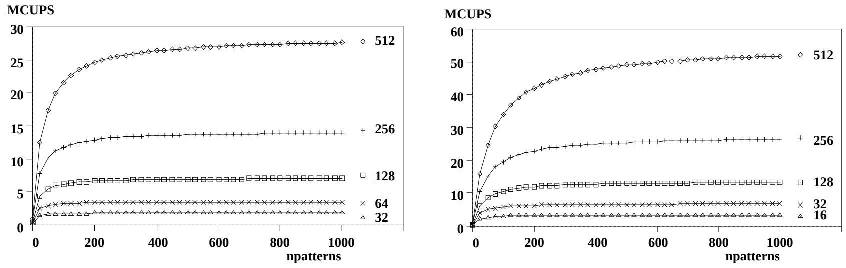
The performance expressed in MCUPS can be calculated by dividing the number of connections that are updated by the total time (in μseconds):

$$MCUPS(P, p, n, w) = \frac{p \cdot w}{T(P, p, n, w)} \tag{6}$$

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. Figure 1(a) plots the measured performance expressed in MCUPS for backpropagation networks, with problem sizes from 1 to 1000 patterns per processor. The performance is predicted correctly within 4%-9% (which corresponds to the deviations of the predicted values for the communication and calculation times). We have measured that even for relatively small (1 Kweights) networks, the prediction appears to be correct within 9%. Figure 1(b) depicts the measured performance expressed in MCUPS for Kohonen networks. Because the measured and expected calculation times match better compared to the backpropagation network, the performance can be predicted with even higher precision ($< 1\%$).

The expected speedup is computed as:

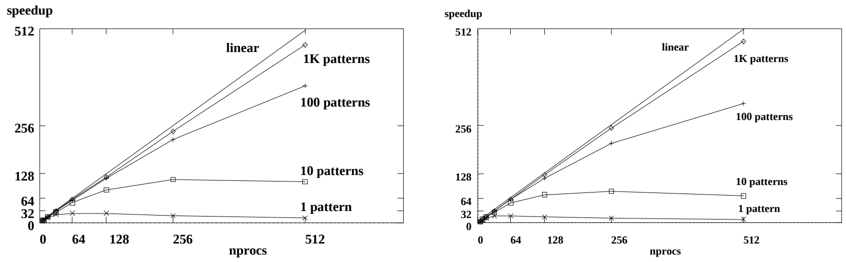
$$S(P, p, n, w) = T(1, p, n, w)/T(P, p, n, w)$$



(a) *Performance for backprop.* (b) *Performance for Kohonen.*

Fig. 1. Performance of different GCel grids.

The obtained speedups are depicted in the figures 2(a) and 2(b) and agree within 1%-9% with the predicted values. 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. Note also that only for small problems (less than 10 patterns per processor) the speedup limit is reached.



(a) *Speedups for backprop.* (b) *Speedups for Kohonen.*

Fig. 2. Speedups of different GCel grids.

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 speedups. As can be observed from figures 1(a) and 1(b), 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.

3 Conclusions

We have presented a method for predicting the performance of a MIMD execution platform for parallel neural network simulations. The model used is based on only a small number of parameters that depend on the execution platform:

the calculation times, the accumulation times and the communication times. These parameters can be measured on one processor and one transputer link. We were able to quantitatively validate the model for large processor networks on a GCel-512. The model appeared to be correct within about 9% and allows to predict the performance, speedup and efficiency for larger processor networks.

The method presented here is general purpose in the sense that it can also be used for other applications. For any application for which the overall costs can be estimated by a small number of calculation and communication benchmarks, the method can be applied. For example many parallel image processing applications show this behaviour. We have also applied our method for neural networks using other decomposition techniques than dataset decomposition [7].

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 [4], one could roughly say that the T9000 is about 8 times faster than the T805 and has about 6 times higher bandwidth. Furthermore, we expect that when exploiting the new C104 communication chips, it will be cheaper to use all-to-all communications, thus reducing the communication times. From equation (1) it can be expected that the performance will at least increase by a factor 6 to 8, up to 200 and 400 MCUPS for backpropagation and Kohonen networks respectively. This means that transputer systems based on the T9000 and C104, are even better suited for parallel neural network simulations than current ones.

References

1. 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.
2. A.J.G. Hey. The Genesis Distributed Memory Benchmarks. *Parallel Computing*, 17(2):1275–1283, 1991.
3. T. Kohonen. *Self-Organization and Associative Memory.* Springer Verlag, Berlin, second edition, 1988.
4. Inmos/SGS-Thomson Microelectronics. The T9000 Transputer Product Overview, 1991.
5. 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.
6. D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.
7. 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.
8. M. Witbrock and M. Zagha. An Implementation of Backpropagation Learning on GF11, a Large SIMD Parallel Computer. *Parallel Computing*, 14:329–346, 1990.