IN-62

43055

Neural Networks and MIMD - multiprocessors

P 17

Jukka Vanhala
Kimmo Kaski

February 1990

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 90.9

NASA Cooperative Agreement Number NCC 2-387

RIACS

Research Institute for Advanced Computer Science
An Institute of the Universities Space Research Association

Neural Networks and MIMD - multiprocessors

Jukka Vanhala
Kimmo Kaski

RIACS Technical Report 90.9

February 1990

Abstract. Two artificial neural network models are compared. They are the **Hopfield neural network** model and the **Sparse Distributed Memory** model. Distributed algorithms for both of them are designed and implemented. The run time characteristics of the algorithms are analyzed theoretically and tested in practise. The storage capacities of the networks are compared. Implementations are done using a **distributed multiprocessor system.**

This work has been supported partly by the Center for Technological Development in Finland (TEKES) and at RIACS by NASA Cooperative Agreement NCC2-387. Kaski spent a part of his sabbatical year from Tampere University of Technology, Finland, as a visitor at RIACS.

Neural Networks
and
MIMD - multiprocessors

Jukka Vanhala and Kimmo Kaski

Tampere University of Technology
Microelectronics Laboratory
P.O.Box 527, SF-33101 Tampere, Finland.

Abstract

Two artificial neural network models are compared. They are the **Hopfield neural network** model and the **Sparse Distributed Memory** model. Distributed algorithms for both of them are designed and implemented. The run time characteristics of the algorithms are analyzed theoretically and tested in practise. The storage capacities of the networks are compared. Implementations are done using a **distributed multiprocessor system.**

1. Introduction

Artificial neural network models are originated from theoretical neurobiology but they serve as practical tools for computing. Neural networks are highly connected systems consisting of simple threshold units. Their inherent parallelism, fault tolerance and learning ability makes them very useful when the conventional methods fail or perform poorly. On the other hand their massive parallelism and high connectivity also makes them hard to implement on traditional computer architectures. To make it a bit easier we have tried to analyze some aspects of running neural networks on a distributed multiprocessor.

This paper compares two implementations of neural network models, namely the Sparse Distributed Memory model [Kanerva -88] and the Hopfield neural network model [Hopfield -82]. The mathematical formulations for both of these models are shown in reference [Keeler -86]. The Sparse Distributed Memory model (SDM) is in many respects comparable to the Hopfield neural network model. The ideas behind these models are quite different but the resulting behavior is very similar. Both of them can function as an autoassociative memory and both utilize the Hebbian learning rule. Both network algorithms are implemented on a distributed Transputer based multiprocessor and their behavior is analyzed.

1.1 Hopfield model

Hopfield network is a fully connected network with a symmetric weight matrix. It can be thought as having two layers, input and output. If the output is fed back to the input, the network becomes effectively a one layer network.

Initial configuration is loaded into the network and it is then released to evolve freely. After the network has converged to a stationary state, the output can be read out. Each neuron in the network decides its state using the following equations:

$$v_i(t) = \sum_j C_{ij} s_j(t)$$

$$s_i(t') = sign(v_i(t))$$

Where s_i is the state of the i th neuron, either 1 or -1 and v_i denotes the potential induced to i th neuron by all other neurons. There are two basic strategies to calculate $v_i(t)$ and $s_i(t')$ values. The classical way is to pick one neuron at random and solve both equations for it updating the new state of the neuron immediately. The method is referred to as asynchronous updating. The other strategy is to calculate first potentials the v_i for each neuron and then update the new state values $s_i(t')$ simultaneously. This is referred to as synchronous updating. The later approach is more suitable for implementation in a distributed environment as will be seen later.

1.2 SDM model

Sparse distributed memory can be described either using concepts of digital computers or using concepts of neural networks. Viewed from outside, SDM has address and data buses and read-write control. The data bus and the read-write control function similarly to a conventional random access memory (RAM) system. The address space of a RAM system is small, i.e. of the order of 2^{16} to 2^{32}. On the other hand the address space of SDM may be very large, for instance of the order of 2^{1000}. It is clear that this much of actual memory can not be implemented (since 2^{1000} exceeds the number of atoms in the universe). In SDM, the huge address space is covered sparsely with randomly chosen addresses. When data is written to SDM, its address is very likely pointing to a nonexistent memory location. Thus the written data is distributed to those actual memory locations whose addresses differ from the desired address by only a small amount. When the data is written into a conventional RAM, the old data is lost. In SDM, new data will be superimposed with the old data in memory. Thus every bit in the memory has to be a counter rather than a one-bit register. If the address space is 2^{1000} and the SDM system implements say 2^{16} storage locations, these cover one address in 2^{984}.

In figure 1, the internal structure of a possible implementation of SDM is shown [Keeler - 86]. There are two matrices A and C. A is used to store the addresses of those memory locations that are actually implemented. C is a matrix of counters which stores the information written into SDM. When the memory is accessed, the address vector $\boldsymbol{a}$ is compared with storage addresses. The selector vector $\boldsymbol{s}$ has elements set to one corresponding to each address in matrix A which is close enough to the original address in terms of the Hamming distance. When doing a write operation, data ***d*** is accumulated at every counter location in matrix C as selected by ***s***. Bits which are one in ***d***, increment the value of the counter, and bits with value zero decrement it. Reading from SDM proceeds much the same way as writing. The addresses are compared and the selector is formed. Then every selected memory word is summed together and tresholded with zero to give the output data ***d'***. Thus the output has one-bits for non-negative values and zero-bits for negative values of the sum.

Although the idea of SDM can probably be explained most clearly using conventional computer concepts, it can also be viewed as a three layer feed forward network. If the address length is m, the data length is d and the number of actual storage locations is p, SDM is an m-p-d network, i.e. it has m nodes in the input, p nodes in the hidden and d nodes in the output layer. This view of the model shows the close relationship between SDM and the models normally denoted as neural networks.

2. Distributed algorithms

The most critical problem in implementing neural networks on distributed computers or as a matter of fact on silicon is the communication overhead. Since the network is (often) fully connected, every time a neuron decides to chance its state the new value has to be distributed to every other node. The communication delay between processing elements makes the convergence of the network unsure or at least has a potential effect on choosing the final configuration the network will eventually converge to. It is also very difficult to build a system with a high number (m) of nodes and full connectivity since the number of connections grows as m^2. Thus, if the network is implemented in a straightforward manner, the communication channels have to be multiplexed which still slows down the operation.

We have implemented distributed algorithms for both network models. The algorithm for the Hopfield network is a well known solution for calculating systems with long range interactions, namely the n-body problem algorithm. The algorithm for SDM is derived from the model presented in figure 1.

2.1 Hopfield algorithm

Implementing a Hopfield type network using asynchronous updating gives rise to communication problems. After each update the new state value must be distributed to all other processor. In fact this forces the system to proceed sequentially without any profit from multiple processing elements. Processors may be allowed to calculate new states for neurons they are allocated using old state values of foreign neurons which have not yet been updated due to communication delay. This does not reduce the communication but lets the processors run (very probably doing wrong updates) which would further slow down the operation.

On the other hand if the network implementation is allowed to use synchronous updating, it can be implemented efficiently on a multiprocessor machine by the n-body problem algorithm. Processors are connected to form a ring topology. Another commonly used topology is a hypercube, which can also be used since a ring topology can be embedded in hypercube topology. Each processor is assigned a group of neurons (assuming that the number of neurons in the network is greater than the number of processors) and their corresponding

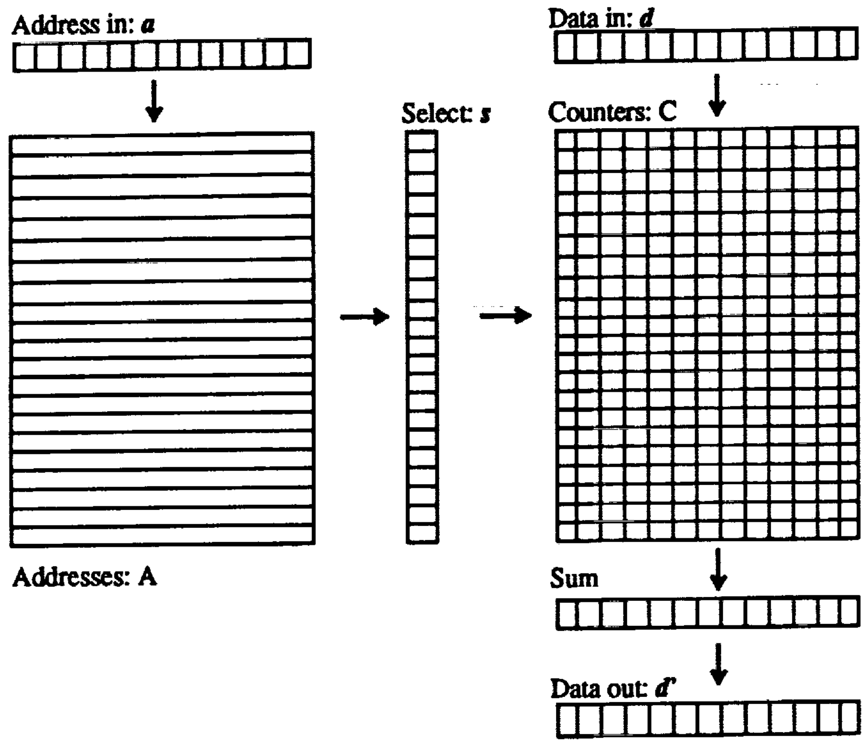

Figure 1. Internal structure of a possible implementation of the SDM.

weight matrix elements. For each neuron there is a packet that travels around the ring and accumulates the potential created by other neurons. During a visit the processor calculates the effect of its neurons on the visitor. After visiting all other neurons, the packet arrives to the neuron it is assigned to. From the accumulated potential the neuron can decide its next state. Since all neurons use the old state values of all other neurons, the whole network is updated synchronously.

This does not reduce the communication but keeps the processors occupied. It should be said that this idea does not faithfully follw the dynamics through the state space, as described by the Master equation. On the other hand near the stationary sates of the network this updating scheme should be in average sufficiently accurate.

2.2 SDM algorithm

Since SDM can be described without referring to a network model with high connectivity, it seems to lack the communication overhead problems. Every operation and data structure is well localized. The address matrix A, selector vector *s* and the counter matrix C can be sliced horizontally without difficulty. The only necessary communication is to distribute the initial address and data (only address in read operation) and to collect the sums from counters. The matrix A can also be implemented as a pseudo-random number generator which gives the same sequence of location addresses every time the selector *s* is calculated.

3. Storage requirements

The information in the Hopfield model is stored in a symmetric m*m weight matrix W where m is the length of the input pattern. As W is symmetric, it would suffice in theory to store only half of the matrix. We have not used this optimization since it would impose more complexities to the distributed algorithm. Weight values are obtained by Hebbian learning rule. A weight value w_{ij} is increment for every pattern that has bits i and j in the same state. If these bits differ the weight value is decrement. For random patterns (which we have used) there is no correlation between bits and thus the weight values tend to be small. Even for our largest test case the maximum number of patterns is ~100. If some of the bits in patterns were clamped to say one, this would generate a weight value of ~100. Thus we have chosen to use 8 bit bytes to present weights. This gives us a range -128 ... 127. We have not encountered overflows.

The weight matrix has been divided between the processors. There is thus no redundant information stored (other than the symmetrical parts of the matrix). The processor that "owns" neurons s_{i1} ... s_{i2} stores the weight matrix rows w_{i1} ... w_{i2}. The processor is then always capable of calculating the effect of its own neurons to any other neuron.

SDM stores information in a m*p table. p equals m for an SDM system with about the same storage capacity as a Hopfield network with input size m. In this case the information density is the same for both network types. The reasoning for the implementation of a Hopfield type network to use 8 bit counters also holds for SDM.

The address matrix A and the counter matrix C have been divided in equally sized parts to each processor. Although this method is not mandatory and we in fact lose Mbytes of memory due to imbalance in our system memory configuration, it balances the processing load between nodes. The location address matrix A is stored in packed format, i.e. a 1024 bit address occupies 32 words of 32 bit memory. If we take matrix A into account when calculating information densities we get ~10% lower result (if p = m).

4. Capacity

A definition for the capacity of SDM is, according to [Kanerva -88], the size of the data set for which the probability of reading a given pattern correctly from the address it was written in is 0.5. As the capacity scales linearly with the number of storage locations p, it is convenient to give a capacity factor c which gives the capacity as a multiple of the number of storage locations. Factor c depends on the length of patterns and for our test cases it is c=0.13 for m=256, c=0.11 for m=512 and c=0.098 for m=1024.

The capacity of the Hopfield network has been studied in literature extensively both experimentally and with mathematical rigor. Reference [McEliece -87] gives the upper limit m/(2logm) for storing patterns of size m given that *most* of the written patterns must be recalled correctly. In our case this gives c=0.090 for m=256, c=0.080 for m=512 and c=0.072 for m=1024. This is lower than in the case of SDM. The difference might be due to the fact that [McEliece -87] requires that every written pattern has the basin of attraction circle non-zero whereas [Kanerva -88] lets it to become a point in the address space. These differences are small and it is justifiable to expect a capacity of about c=0.1 ... c=0.15 for both systems depending on the actual set of patterns and the requirements for the correctness of recalled patterns.

If a pattern is not recalled correctly, the read chain may either converge to a stable but spurious state or diverge to a chaotic wandering trough the space with no fixed destination. Since Hopfield type network relies on the analogy to a physical terms of energy, it will always converge to a fixed state, be it real or spurious. In SDM the error is often of the latter chaotic type. Convergence in the Hopfield network will quite often give a result that is very near the perfect result and depending on the case the slightly erroneous answer may be usable. On the other hand if the network always gives an answer it is hard to say anything about the quality of the answer. In SDM the chaotic behavior is easy to distinguish from the fast convergence to a correct answer. This gives a way to differ between right and wrong answers.

5. Run time performance

The most interesting performance measure of a multiprocessor implementation is the speedup factor [Fox -88]

$$S(N) = \frac{T(1)}{T(N)}$$

where S(N) is the speedup factor depending upon the number of processors N and T(N) is the run-time of a calculation in an N processor system. T(1) is of course the run time on a sequential machine. S(N) tells what is the average utilization of the N processors in the system. For example if S(N) = 9 for a 10 node machine the processors are working on the problem 90% of the time and communicating and processing "house keeping" information 10% of the time.

he speedup factor can also be given by the equation

$$S(N) = \frac{N}{1+f_c}$$

where f_c is the fractional communication overhead. It can be thought as the fraction of the total run time spent on communication. It can be written

$$f_c = \frac{c}{n^{d_p^{-1}}} \frac{t_c}{t_w}$$

Here c is a constant that depends on the characteristics of the algorithm and is normally in the range 0.1 ... 10. d_p is the dimensionality of the problem. n is the grain size i.e. the amount of work assigned to each node. t_c and t_w are constants typical to each computer system denoting the times to communicate one word of data from one processor to another (t_c) and to make one typical calculation (t_w). For example if S(N) = 9 and N = 10 then fc = N/S(N) - 1 = 10/9 - 1 = 0.11.

5.1 Hopfield network

In a Hopfield network, the inner product of the weight matrix row C_i and state vector S must be computed for one update of neuron i. This results in the computational complexity of p multiplications and p additions, in which p is the number of neurons in the network. The threshold function must also be computed, thus giving the total complexity of O(2p+1). For one synchronous update for the whole network, p neurons calculate their next state giving the complexity $O(2p^2+p)$. If asynchronous updating is used, it can be thought that one update for the whole network involves p randomly chosen neurons to be updated. This yields the same complexities as above. Using modified n-body algorithm, the processor network with N processors passes N data packets of size p/N in N steps. Thus the communication has the complexity of O(N p/N N) = O(Np).

The combined run time for the Hopfield network algorithm is

$$T(N) = N(t_p + t_r)$$

where t_p is the time for calculating the potentials and t_r is the time for rotating data. The algorithm works in N steps thus the factor N. The calculation time can be given

$$t_p = n^2 t_w$$

where n = p/N is the number of neurons assigned to each processor and t_w is the time for calculating one synaptic interaction. Rotating the cumulative values in the ring takes

$$t_r = n t_c$$

This gives the fractional communication overhead of

$$f_c = \frac{n t_c}{n^2 t_w} = \frac{1}{n} \frac{t_c}{t_w}$$

This shows that the dimensionality of the problem is $d_p = 1$, as it should be for long range interaction problems.

The sequential parts of the implementation are not considered above. It might be relevant, since if the sequential part of the algorithm takes a fraction s of the total run time on a sequential machine, then it is impossible to achieve greater speedup factor than s^{-1}. This is the famous Amdahl's law. The solution to this problem is in that the bulk of the computation supersedes almost completely the sequential parts of a typical scientific problem with even a modest size.

5.2 SDM

In order to retrieve data from SDM, p hamming distances, H*d additions and d threshold functions must be computed. Here p is the number of actual storage locations, d is the length of the address and the data, and H is the number of storage locations that will be selected on average. Thus the overall complexity is O(p+Hd+d). In a fair comparison with the Hopfield network d = p and H equals the square root of p giving the complexity $O(dp+p^{3/2}+d) = O(p^2+p^{3/2}+p)$ which is the same as for the Hopfield network.

The communication structure of the SDM implementation is very simple. If we again forget the sequential part, there is no communication at all! This would give the problem dimensionality dp = 0 and thus zero fractional communication overhead and linear speedup with any number of N and value of n. SDM differs from the Hopfield network in that where the master processor of the latter algorithm collects only ready answers from the network, the master processor of SDM still has to add and threshold all results obtained from other processors. It seems thus fair to consider this as a part of the SDM algorithm. With sequential parts added we get a decrease in performance by Amdahl's law. Thus the run time is

$$T(N) = 2Nmt_c + Nmt_t + nt_w = Nm(2t_c + t_t) + nt_w$$

The first term of the sum denotes the time required for sending the initial configuration to every processor and for collecting the results back to master processor. The second term is the sequential part of the algorithm where subtotals of length m from N processors are handled. The third term denotes the time taken by processing the n = p/N elements stored on each node. The fractional (communication) overhead is

$$f_c = Nm\frac{2t_c + t_t}{nt_w}$$

Although f_c depends heavily upon the number of processors N, the overhead is small as long as p>>N. As we have compared Hopfield type network with SDM, m has been chosen to be equal to square root of the total number of neurons p. This does not have to be the case and normally p is limited by the amount of available memory on each processor. In our system we are able to store ~2^{20} elements per node. Contrasted to a typical number of processors (~2^5) and size of m (~2^{10}), the overhead is still of the order of 2^{-4} ... 2^{-5}.

To avoid the sequential part of the algorithm, one could slice the counter matrix of SDM vertically. Then every node has a slice of every word in the memory. This has the drawback of generating the location address matrix to every node. One solution is to generate location addresses by a pseudo-random number generator simultaneously with the Hamming distance calculation. In this way only the seed for the generator has to be stored. A good random number generator may be too time consuming to run as a part of a simulation on a general purpose computer, but as a part of a VLSI implementation the method could be feasible.

6. Simulations

Simulation studies of the Hopfield type network and SDM are shown. Implementations were designed so that the results will be comparable. The same processor network topology is used for both network models and they are used to solve the same problem with same training and test sets.

6.1 Environment

Simulations were run on a 10 node Transputer multiprocessor. Each node has a T800-processor and at least 1 Mbyte of memory. The system has been installed in a "dummy" PC/AT chassis from which it only gets its main power. This unit is connected via one INMOS link to a Transputer card installed in another PC/AT which is used as a console and a file server. All 10 nodes in the machine are connected in a pipeline. The first node (the one connected to the PC/AT) is the master node having the reset and error lines in its control. A ring topology would be better in terms of run time performance because the Hopfield algorithm has ring communication topology.

The only software tool used was Logical Systems Transputer Toolset which has a C compiler with normal back end tools and run time libraries. A general purpose message passing communication kernel was written and used for developing the network implementations. Since all communication is delayed by the message passing routines, the run times are much longer than they would have been if raw channel communication had been used instead. On the other hand the system development has been much easier and faster since the use of the message passing routines have hidden the topology of the machine and have made debugging possible. We have tried to make relevant analysis of the run times by substracting the effects of delayed communication.

6.2 Training sets

The training was done using a set of patterns generated with a linear shift register random number generator. The same training set was used for both network types in all simulation runs. The quality of the random numbers seems to have a great effect on the performance of the networks. For example the random number generator rand() which is included in the C libraries gave much worse results in terms of capacity.

6.3 Capacity

In order to test the capacity of the networks, the patterns in the training set were stored in the network one at a time. After each write operation all previous examples in the training set were tested by reading from their write address. The patterns not recalled correctly were counted. Also patterns differing more than 1% from the original were counted to give some indication of the quality of the returned pattern. The results of the simulation runs are shown in the figures 2, 3 and 4, which give the percentage of correctly recalled patterns as a function of the number of written patterns. n is the size of the network.

For m = 256 both networks exceed clearly their theoretical capacity factors. SDM should give 50% correct patterns at the limit of 33 written patterns. The Hopfield network should saturate at 22 patterns. Both networks almost double the limit. Same kind of behavior is also apparent in the two other cases but with smaller marginal.

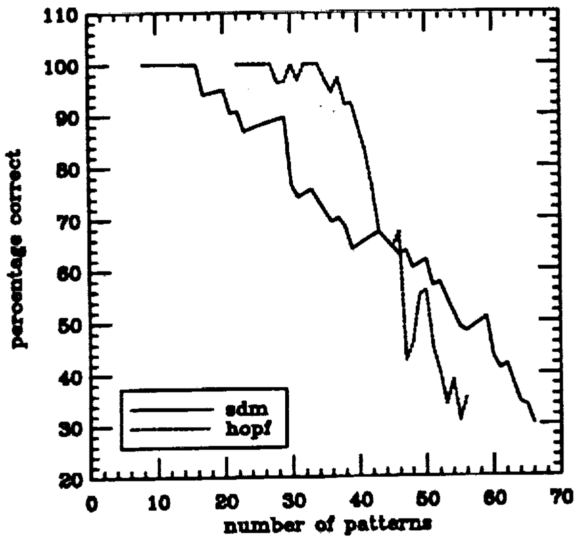

Figure 2. Capacity, m=256

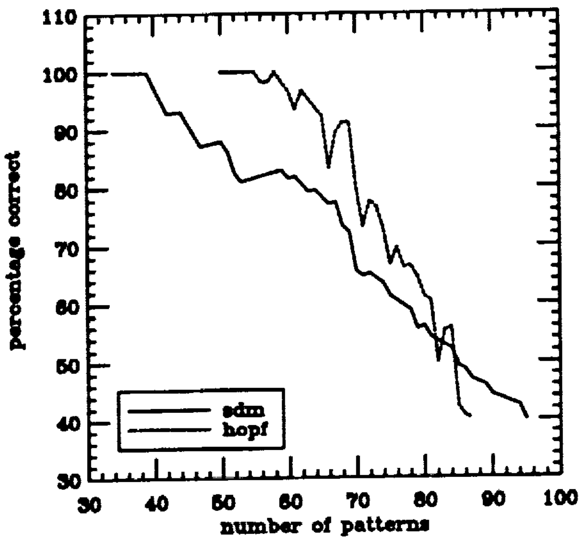

Figure 3. Capacity, m=512

For the Hopfield network, notably only about 5% of the incorrectly recalled patterns differed more than 1% from the original at the limit, where 50% of the patterns were recalled correctly (for n = 1024, not shown in the diagram). For SDM, almost all incorrectly recognized patterns diverged and only a minimal fraction converged to a stable state near the correct answer.

SDM is very sensitive to the cutoff limit parameter as is shown in figures 6 and 7. They present the relative performance of the SDM network with 256 storage locations for some values of the cutoff parameter. If the value is less than optimal (here 111), the network has difficulties to learn even a small number of patterns correctly. On the other hand, since ev-

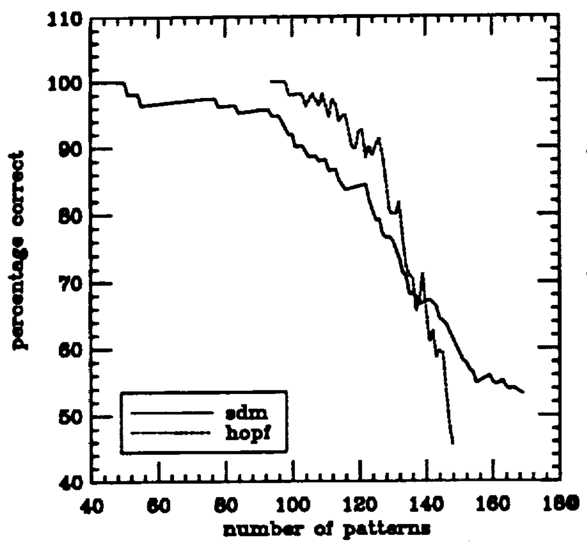

Figure 4. Capacity, m=1024

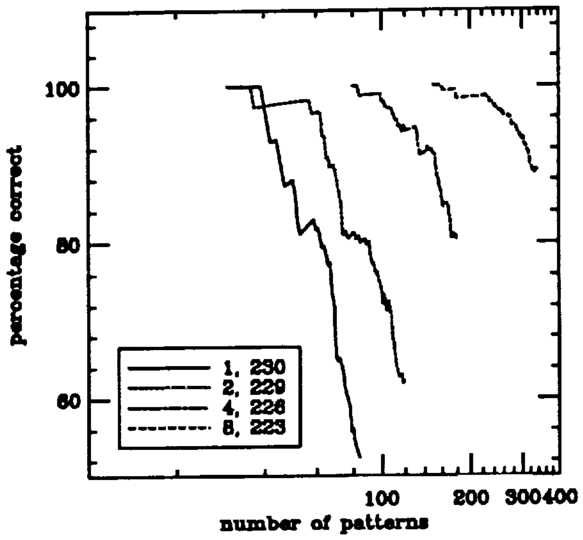

Figure 5. Scaling of capacity, legend shows the size multiplier and the cutoff value (m=512).

ery pattern is written in fewer storage locations, they do not overlap so easily. Thus the network is able to store a great number of patterns although with smaller probability to get them right. This is easy to see in the diagrams as the solid line denoting the low cutoff value drops below 100 % in the very beginning but has a smaller decrease in the end. If the cutoff parameter is too high, the SDM network behaves badly in the beginning but has very steep slope in the end.

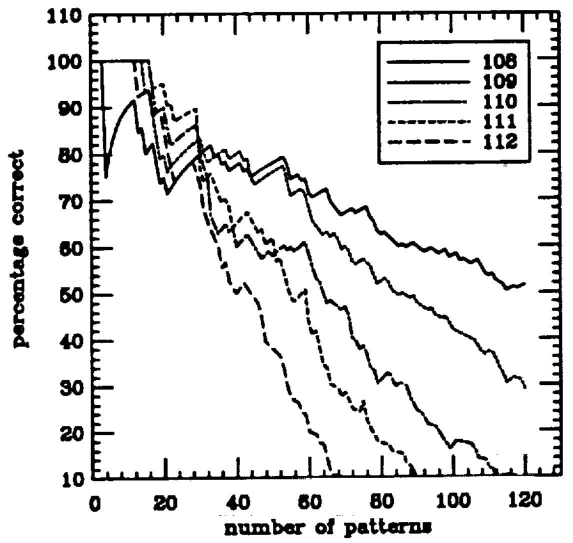

Figure 6. The effect of cutoff limit

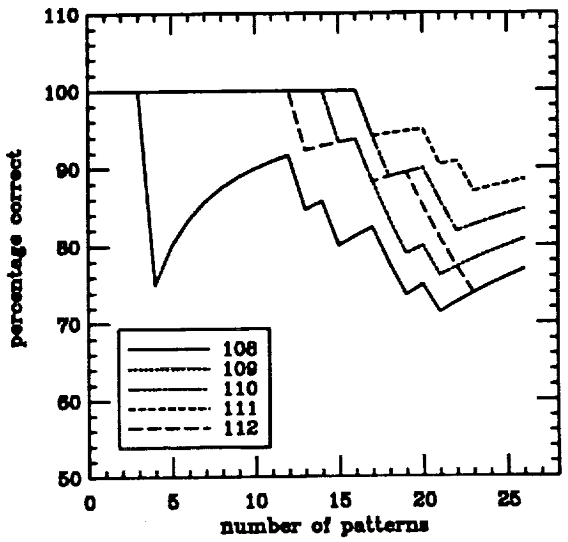

Figure 7. The effect of cutoff limit magnified

Figure 5 shows how the capacity of SDM scales linearly with the number of storage locations. There are four runs with the compoud amount of storage multiplied by a factor of 2 in each run. As the x-scale is logarithmic, the lines should be equally spaced which is the case here.

6.4 Run time performance

We measured the elapsed run time for both networks with two problem sizes, see Table 1. The speedup factor S(N) and scaled run times are shown in figures 8 and 9 for network sizes 1024 and 2048, respectively.

Number of processors	Size of the network	Hopfield	SDM
1	1024	378	22.7
2		193	14.4
4		137	11.1
8		88	9.84
1	2048	1512	90.8
3			37.2
4		530	31.0
6			24.7
8		275	22.0

Table 1. Run times of the simulations (in seconds for 10 store and 50 retrieve operations)

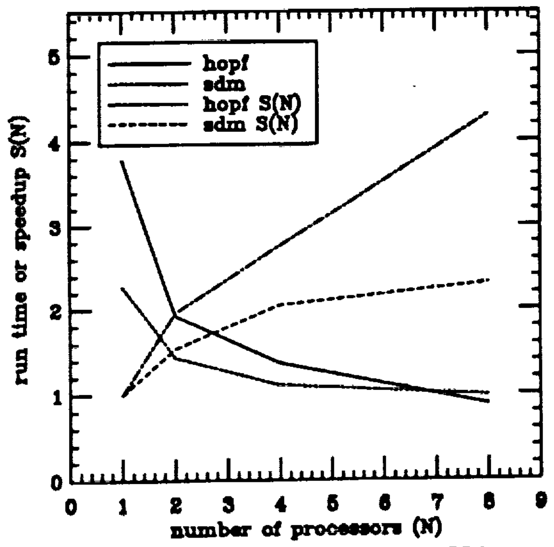

Figure 8. Run times and speedup, m=1024

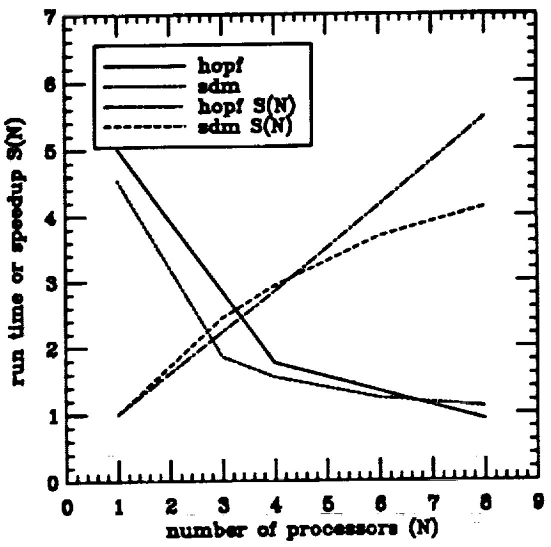

Figure 9. Run time and speedup, m=2048

We have measured time constants for communication and calculation approximately. The values are

$$t_c = 150us$$
$$t_w = 5us$$

Especially the communication time t_c varied much, almost an order of a magnitude. This is because the processor network topology of our simulation environment is far from optimal. The calculation time has a smaller variance. Note that the run times for SDM are about 10 times shorter than for the Hopfield network. On the other hand, the curves show that with the tested network sizes, the speedup of SDM starts to level off while the Hopfield still gives linear speedup. The fractional (communication) overhead for both networks is given in table 2.

Table 2 and the diagrams clearly show the effect of grain size. m = 2048 seems to be enough for the Hopfield algorithm beyond the tested 8 processors. For SDM, the speedup curve bends already at about N=3. This implies that the SDM algorithm is lighter to calculate and the grain size should be increased further.

		equation		diagram	
		Hopf	SDM	Hopf	SDM
N = 4	m = 1024	0.24	0.96	0.45	0.95
	m = 2048	0.12	0.48	0.41	0.37
N = 8	m = 1024	0.47	3.80	0.86	2.46
	m = 2048	0.23	1.89	0.46	0.94

Table 2. Fractional communication overhead.

7. Discussion

Both of the networks, the Hopfield network and SDM, gave expected results in terms of capacity for correctly recalled patterns. The difference in run times were bigger than we thought. Also we expected better speedup factors for the SDM model.

The run time tests were difficult to do using our current processor network topology. The main reason for this reflects in the inbalance of time constants t_c and t_w. For a well balanced system these values should be about the same rather than differ with a factor of 30.

Although the Hopfield model is not very usable in real world systems its behavior serves as a good basis for testing the performance of other network models. The behavior of SDM was found to be very similar to the behavior of the Hopfield network. The possibility of scaling the capacity of SDM makes it more suitable for practical applications.

8. Acknowledgements

This work has been supported partly by the Center for Technological Development in Finland (TEKES) and NASA Ames Research Center, RIACS (Cooperative Agreement Number NCC 2-387). Kimmo Kaski would also like thank RIACS' SDM-group for its kind hospitality and enlightening discussions.

References:

[Hopfield -82] J.J.Hopfield: "Neural Networks and Physical Systems with Emergent Collective Computing Abilities", Proc. Nat. Acad. Sci. U.S., Vol. 79, Apr. 1982.

[Kanerva -88] P.Kanerva: "Sparse Distributed Memory", The MIT Press, Cambridge Massachusetts, 1988.

[Keeler -86] J.D.Keeler: "Comparison Between Sparsely Distributed Memory and Hopfield-type Neural Network Models", RIACS Technical Report 86.31, NASA Ames Research Center, Dec. 1986.

[McEliece -87] R.J.McEliece, E.C.Posner, E.R.Rodemich, S.S.Venkatesh: "The Capacity of the Hopfield Associative Memory", IEEE Transactions on Information Theory, Vol. IT-33, No.4, Jul 1987.

[Fox -88] G.Fox, M.Johnson, G.Lyzenga, S.Otto, J.Salmon, D.Walker: "Solving Problems on Concurrent Processors, Volume 1", Prentice-Hall Englewood Cliffs, New Jersey, 1988.