Communication Parameter Tests and Parallel Back Propagation Algorithms on iPSC/2 Hypercube Multiprocessor

Brian K. Mak *
Speech Technology Laboratory,
3888 State Street,
Santa Barbara, CA 93105

Ömer Eğecioğlu
Department of Computer Science,
University of California,
Santa Barbara, CA 93106

Abstract

The communication complexity on Intel's second generation iPSC/2 hypercube and its effect on parallelization of Back Propagation type training algorithms for neural networks are explored. On iPSC/2, different broadcasting methods are tested and three inter-node communication schemes are evaluated based on their performance on vector addition. These communication schemes are then utilized on parallel versions of the Back Propagation training algorithm. The performance of the resulting parallel variants of Back Propagation are analyzed using two medium size problems: vowel classification and English text-to-speech conversion (NETtalk data).

Introduction

Massive parallelism is believed to be essential for achieving human-like performance in fields such as speech and image recognition. Simulation of massive parallelism can be considered in at least three contexts: parallelism in modeling, computation, and algorithm design. Artificial Parallel Distributed Processing (PDP) neural network models attempt to encapsulate the model parallelism. On the other hand, computation parallelism can now be attained in many new parallel machines, such as Intel's iPSC/2 hypercube multiprocessor.

In this paper we concentrate on utilizing different communication mechanisms of Intel's second generation hypercube, iPSC/2 (Intel's Personal Supercomputer generation 2) to parallelize the Back Propagation (BP) training algorithm and its corresponding Steepest Descent (SD) version for feed-forward neural networks. First, different broadcasting methods are tested and three inter-node communication schemes are evaluated on iPSC/2, based on their performance on vector addition:

- Alternate Direction Exchange (ADE),
- Hypercube Gathering (HG),
- Nearest Neighbor Communication (NNC).

Communication parameters such as start-up time and data transmission rates for iPSC/2 are determined. Each of inter-node communication schemes is then used in parallelized variations of BP and SD. The design issues of the resulting alternate methods center around enhancing computational parallelism, providing relaxation, determining the frequency of required updates, and minimizing communication costs on message-passing multiprocessors.

A summary of the performance of the resulting parallel algorithms on vowel classification and English text-to-speech conversion is given.

Hypercube Communication

It is well known that running an application on a message-passing parallel machine has to take into account message-passing costs, as the processors must communicate with one another to co-ordinate activities. Theoretically, the communication time may be expressed as:

$$T = \beta + \sigma L \tag{1}$$

where T = total communication time in ms;
β = communication start-up time in ms,
σ = data transmission rate in $\mu s/byte$; and,
L = size of message body in $bytes$.

If L is small, $\beta \gg L\sigma$, and it is wiser to pack several small messages into one (if the application permits)

*Research partially supported by fundings from UCSB Micro Proposal No. 615287; Agency Award No. 583002 (ESL Incorporated).

0-8186-2113-3/90/0000/1353$01.00 © 1990 IEEE

and send only once. Some useful and well known algorithms for global communication in a hypercube are described below:

1. **Hypercube Broadcast**

   Aim: To send the same piece of data of size L from the node labeled 0^d to all other nodes in a d-cube.

   Algorithm: Broadcast from root down to the leaves in a hypercube spanning tree.

   STEP 1 Node 0^d sends data to $0^{d-1}1$ of level 1 in the spanning tree.

   STEP 2 For $k = 2, 3, \ldots, d$, all nodes at levels less than k send simultaneously the data received to their own children at level k.

   Theoretical Time: $d(\beta + L\sigma)$ in d steps.

2. **Hypercube Gathering**

   Aim: All nodes send their own piece of data to a particular node labeled 0^d in a d-cube.

   Algorithm: Gather from the leaves up the hypercube spanning tree to the root.

   STEP 1 For $k = d, d-1, \ldots, 1$, each node at level k, labeled $0^{d-k}1a_{k-1}$ where a_{k-1} represents any $(k-1)$-bit binary number, sends its own accumulated data to its parent labeled $0^{d-k+1}a_{k-1}$.

   Theoretical Time: $d\beta + (2^d - 1)L\sigma$ in d steps, assuming that all original data owned by each node are of size L.

3. **Alternate Direction Exchange**

   Aim: All nodes exchange their own local data with all the others so that finally every node has all the data.

   Algorithm: Split the d-cube into two $(d-1)$-subcubes in each of the d directions successively. During each split, each node of the two subcubes exchanges its accumulated data with its counterpart in the other subcube.

   STEP 1 For $k = 1, 2, \ldots, d$, a node whose kth bit is 1, exchanges its accumulated data with its opposite node — the node with the same label except that its kth bit is 0. In other words each node labeled $a_{d-k}1a_{k-1}$ where a_j denotes a j-bit binary number, sends its accumulated data to its counterpart labeled $a_{d-k}0a_{k-1}$.

   Theoretical Time: $2[d\beta + (2^d - 1)L\sigma]$ in d steps, assuming that all original data owned by each node are of size L.

Refer to [SS88], [SS85], and [BT89] for detailed description of a number of other communication methods and topological properties of hypercubes.

Evaluation of Intel's iPSC/2 Hypercube Communication Parameters

The main distinction between iPSC/2 and its ancestor iPSC/1 is its *Direct-Connect Routing System* (See [Int88] for details of iPSC/2 architecture). Briefly, in iPSC/2 each node has its own Direct-Connect routing module (DCM) which routes and relays messages throughout the system, freeing the node processor for local processing. The only time that a node processor is involved with communication is at the source (initiation) or the destination (reception) of the message. Each DCM supports 8 bidirectional serial communication channels. Each channel is independent from the others, and has a bandwidth of 2.8Mbytes/sec in each direction. It may route up to 8 messages, and simultaneous transmission and reception of messages is possible. The DCM also provides almost uniform message latency for communication between all nodes (neighboring or non-neighboring nodes). This is because the time needed to set the switching mechanism at each intermediate node on a communication channel is only about 1 or 2 μs. Actually it is claimed that multiple node communication takes at most 10% longer than neighboring node communication does [Dun88]. Consequently, the hypercube may be treated like an ensemble of fully interconnected processors for node-to-node communication, but not necessarily for global communication operations [Int88, Dun88].

In the following, the parameters of the iPSC/2 communication channels and the performance of different hypercube communication methods are determined[1].

Echo Test

To find the values of β and σ of the iPSC/2 communication channel, an echo test was performed. Messages of various lengths are sent from node 0 to another node, which then echoes back the received message.

[1] The actual tests were performed on 8/10/89 on the Advanced Computing Facility (ACF) of Cornell Theory Center. The ACF receives funding from New York State and members of the Corporate Research Institute.

The test is looped for 50 times and the average time is taken. This is repeated for 5 times and the median value is noted. A similar procedure is applied to all of the tests in this section.

Observations :

1. For the same Hamming distance node-node communication, the results are similar on cubes of all dimensions ($d = 1$ to $d = 5$). The results for communication in a 5-cube is shown in Table 3 and Graphs 1 and 2.
2. From Graph 1, we see that messages of size less than 100 *bytes* require a fixed amount of time. For message size above 100 *bytes* the expected linear relationship between T and L is validated. The values of β and σ for different Hamming distance node-node communication are found by linear regression. These are summarized in Table 4.
3. These results indicate that we have a fixed data transmission rate of $0.3593\mu s/byte$, and a varying communication start-up time. A linear relationship between the start-up time and Hamming distance is observed (Graph 3) with a correlation coefficient of 0.9981 as:

$$\beta = 0.6744 + 0.03123 * H(ms) \quad . \qquad (2)$$

   Thus each additional hop requires $0.03123ms$ for routing.
4. Because of the small magnitude of the delay in multi-hop communication, 5-hop communication of a $1Kbyte$ message requires only $\approx$10% longer than 1-hop communication of the same message (the longer the message, the smaller the percentage). Thus Intel's claim that for most applications, the nodes in iPSC/2 hypercube may be considered as fully connected from the point of view of node-to-node communication is justified.
5. Our results are very close to those cited in [Dun88].

Broadcast Tests

As many parallel applications require broadcasting a piece of message from one node to all others, e.g. sending an initial condition to all processors, it is useful to find an optimal broadcasting method. Three broadcast tests were performed on iPSC/2:

- using the Build-In Broadcast Function (BIBF);
- using the Hypercube Broadcast (HB) algorithm; and,
- Sequentially Sending (SS) the message to all nodes in the hypercube.

Node 0 ($= 0^d$) is chosen to broadcast the message, and the farthest node is required to send an acknowledgement of 0 *byte* back before the next round, to prevent any pipelining effect. Here we assume that the farthest node is the last node to receive the message (which may not be absolutely true because of unpredictable communication delay for other nodes.)

The time required for messages of various sizes broadcast in a 5-cube is plotted in Graph 4, while Table 5 shows those results in smaller cubes. These results include the acknowledgement time as well.

Observations :

1. All three methods perform almost equally well in 1-cube broadcast as expected since in this case we are basically sending a 1-hop message. The slightly longer broadcast time required for BIBF and HB can be attributed to their algorithmic sophistication.
2. For broadcasting in a cube of dimension greater than 1, the three methods are ranked in decreasing order of performance as:

   $$BIBF > HB > SS.$$

   BIBF and HB give almost the same results. For a message of $16Kbytes$, HB is only 3% slower. It seems that BIBF is actually implemented with the Hypercube Broadcast algorithm. Heavy communication contention accounts for the poor performance of SS.

Vector Addition Tests

The reverse of broadcasting is message gathering. As an application usually needs to gather information from various nodes to compute some kind of global quantity, a purely gathering test is not that meaningful. Instead we do a floating-point vector addition test by the following three methods:

- the Alternate Direction Exchange (ADE) algorithm;
- the Hypercube Gathering (HG) algorithm; and,
- Random Collecting (RC), in which the messages are sent by all nodes directly to one particular node.

Again node 0 is chosen as the gathering node. Each node sends its own vector to node 0. At the conclusion, at least node 0 will contain the sum of all the vectors. To synchronize the gathering, node 0 first sends a starting signal of 0 *byte* to all the nodes by the Built-In Broadcast Function to initiate the gathering algorithm. The floating-point addition is done

while the gathering algorithm is being executed. Thus the measured time is actually the sum of time spent on floating-point vector initialization, broadcasting, gathering, and floating-point vector addition.

The ADE algorithm is worth testing since the iPSC/2 communication channels are bidirectional, thus nodes may exchange data by sending their own data and receiving data from their counterparts simultaneously.

The results for additions of vectors of various dimensions in a 5-cube are plotted in Graph 5. Table 6 is a summary of all of the results obtained.

Observations :

1. All three methods give similar result for 1-cube vector addition as this involves only the sending of a 1-hop message. ADE requires slightly longer time.
2. For broadcasting in a cube of dimension greater than 1, the three methods are ranked in decreasing order of performance as
   $$HG > ADE > RC.$$
3. ADE performs only slightly poorer than HG. Even for floating-point vectors of dimension 4096 or $16Kbytes$, ADE is only 10% slower. However ADE has two advantages over HG:
   (a) At the conclusion, not only node 0 but all nodes have the total sum.
   (b) In applications where every node needs the sum for further computation, the sum must be broadcast back to each node in HG. However by employing ADE, this step may be avoided. If we add to those results of HG the time needed to broadcast the vector sum back to each node, the adjusted times are greater than those of ADE as shown in Table 6(e).
4. The bidirectional nature of the iPSC/2 communication channels does not seem to be as beneficial as expected.
5. Once again RC suffers from heavy communication contention.

PDP Model and Back Propagation Training

The PDP model that we consider is a multi-layer perceptron-like feed-forward nonlinear network. The bottom layer consists only of input units, while the top layer consists only of output units. There may be more than one layer of *hidden* units. Output of a unit in a lower layer is sent only to unit(s) in higher layer(s). When a training pattern is presented to the input layer, the inputs propagate forward by the *Propagation Rule* and a non-input unit combines all its inputs by means of a logistic activation function to determine its activation value. Its output value is then determined by an identity output function. See [RM86] for further details.

BP is a supervised training scheme. It is a variant of the standard gradient descent method, aiming to minimize the mean square error between desired outputs and actual network outputs when a set of training input patterns and the corresponding target patterns are presented to the network. The cumulative error made is a function of the connection weights, so that the minimization is performed over a very large dimensional space. The need to parallelize the training rule itself arises from the time requirements of gradient-descent based algorithms for neural networks, when the dimension of the underlying space is large (typically several thousand).

BP training involves two phases: a *forward phase* and a *backward phase*. During the forward phase, input signals are propagated forward according to the propagation and activation rules through hidden layers to the output units. Error of an output unit due to the difference between the target output and network output values is computed and then propagated backward to each hidden unit which uses all the error in the output units to compute its own. The so called *generalized delta rule* [RM86] is used to update the connection weights in the backward phase.

In the standard steepest descent method, the weights are updated after each *epoch*, (i.e. one presentation of the whole set of patterns to be learned). However in BP, weights are updated after the presentation of *each individual pattern*. In the neural network folklore, one expects that so far as the learning rate is small, the deviation from the gradient descent method should be small as well.

In the following sections we present a summary of parallel variants of Gradient Descent based training algorithms, along with their performance on two speech related problems. A detailed analysis can be found in [EM90].

Parallelized and Modified Back Propagation Algorithms

The standard BP is modified and parallelized in six different ways, namely: Parallelized

- Standard Back Propagation Method (PSBP);
- Pipelining Back Propagation Method (PPBP);
- Averaging Back Propagation Method (PABP);
- Standard Steepest Descent Method (PSSD);
- Pipelining Steepest Descent Method (PPSD); and,
- Averaging Steepest Descent Method (PASD).

Patterns are assigned to the processors as evenly as possible in all cases.

In PSBP, each node learns its own patterns, and updates its own image of the net after each pattern presentation as in normal BP. At the end of an epoch, the nodes send their own total change of each weight during that epoch to each other by ADE and update the net according to the generalized delta rule.

In PPBP, a node is chosen as the Manager, and an epoch is divided into several, say m (which we will refer to as the *Pipelining Factor*) rounds. Each node subdivides the learning patterns it receives so as to learn as evenly as possible on each round. In each round, the Manager updates the connection weights from the computation results of the other nodes based on the old weights of the previous round, broadcasts the new weights to all nodes, and waits for the new computation results from all other nodes. Simultaneously, each node works on its patterns of that round with the old weights of the net using BP, and sends its own total change of each weight to the Manager using HG.

In PABP, each node learns its own patterns, and updates its own image of the net after each pattern presentation as in BP. At the end of an epoch, however, each node sends its own total change of each weight during that epoch as well as its weights to all nearest neighbors, while receiving theirs. It updates each weight by averaging the original values of the corresponding weights collected from all neighbors (i.e. those values at the beginning of that epoch), and then adding to the average all the corresponding weight changes.

The remaining three algorithms PSSD, PPSD, and PASD are Steepest Descent analogues of PSBP, PPBP, and PABP, respectively.

The hypercube gathering communication mechanism used by each method is summarized in Table 1. It is interesting to note that the communication scheme also affects the locality of information. By using ADE, every node shares its information with each other and has a global view of the network. In HG, a node 'knows' the information that belonges to those children on its subtree of the hypercube spanning tree. The higher a node is in the tree, the more it knows; and the root knows all. By NNC a node only 'knows' information about its nearest neighbors. Intuitively, an algorithm using simpler communication such as NNC saves time spent on communication, but it may take more learning epochs as the nodes do not have the global image of the whole net. This is not the case if a more sophisticated communication method is adopted.

Table 1: Hypercube Gathering Schemes Used by Different Parallelized Methods

METHOD	GATHERING SCHEME
Standard	Alternate Direction Exchange
Pipelining	Hypercube Gathering
Averaging	Nearest Neighbor Communication

Note that except PPSD and PPBP in which one node is chosen as the Manager to update the neural net, each node in all other algorithms executes the same set of operations. Also except PSSD, other algorithms will behave differently with different number of nodes used. For BP variations (PSBP, PPBP, and PABP), the weights are updated after each pattern presentation and different number of nodes will result in different distributions of learning patterns. For PABP and PASD, different number of nodes means different number of neighbors, and this will affect the amount of information a node may acquire during each learning epoch. Note also that some degree of relaxation is embedded in the averaging method as each processor just updates its knowledge based on the information it receives from those available locally around it.

Algorithm Performance Evaluation Tests

Two medium-size problems are used to evaluate the algorithms, using PDP models that consist of a single hidden layer with varying number of hidden units:

- vowel classification,
- speech-to-text conversion (based on the NETtalk experiment).

These problems were chosen as BP is found to perform quite well on speech recognition [EZ87, PMT86, SR86]. The two problems have very different characteristics as summarized in Table 2, and the algorithms are found to behave differently on these two problems. Consequently different criteria of comparison are devised for each one.

In all of the tests conducted, the momentum factor is ignored to reduce the number of possible cases of analysis. Furthermore, all communications are synchronous since it was found that the iPSC/2 does not support asynchronous communication well.

Vowel Classification Test

In this experiment a neural net is trained using BP to classify 9 English steady state vowels[2] in an /h-d/ context [HEW88] uttered by 5 male speakers.

Test Details

- Number of learning patterns: 225.
- Number of input units: 17.
- Number of output units: 9.
- Size of net with 4 hidden units: $4 \times (17+9) = 104$.
- Weights of the neural network are initialized to random real numbers in the range 0.0 to 1.0.
- Terminating criterion: 90% correctness in maximum 10000 epochs.
- Message size: $5.168 Kbytes$ for each of PSBP, PSSD, PPBP and PPSD, and $10.336 Kbytes$ for PABP and PASD.

Remarks

1. Efficiency versus number of processors used: Definition of self-efficiency of a parallel algorithm is modified as follows for analysing standard or averaging variations:

$$E_n = \frac{\frac{T_1}{N_1}}{n \times \frac{T_n}{N_n}} = \frac{1}{n} \times \frac{T_1}{T_n} \times \frac{N_n}{N_1}, \quad n > 1, \quad (3)$$

where E_n = self-efficiency with n processors;
T_1 = processing time with 1 processor;
T_n = processing time with n processors;
N_1 = no. of epochs req'd with 1 processor;
and, N_n = no. of epochs req'd with n processors.

And for the pipelining variations, the following is used:

$$E_n = \frac{1}{(n-1)} \times \frac{T_2}{T_n} \times \frac{N_n}{N_2}, \quad n > 2 \ . \quad (4)$$

2. Comparisons are done on a 'likely' optimal setting under which learning is accomplished with the least number of pattern presentations.

[2] Data came from the B subset of the Speech Technology Laboratory Vowel Database, courtesy of Speech Technology Laboratory – a division of the Panasonic Technologies Inc. of Matsushita in Santa Barbara.

Test Results and Observations

Optimal results for vowel classification test on a 4-cube appear in Table 7 and the efficiency results are given in Table 8. Some observations are as follows:

A. With regard to the optimal results of vowels classification test on a 4-cube:

(1) Each of the standard and pipelining methods requires similar number of epochs to complete learning, regardless which of the two approaches (SD or BP) is used.

(2) PASD works quite well and is comparable to PSBP.

(3) In terms of the number of training epochs required, pipelining method learns faster with a larger pipelining factor.

(4) The performance of the various methods on T/N ratio is in the following ascending order:

standard < pipelining (with appropriate pf) < averaging.

B. With regard to the efficiency results:

(1) In general, efficiency decreases with the dimension of hypercube used.

(2) The efficiency on high dimensional cubes decreases in the following order:

pipelining($pf = 2$) > standard > pipelining($pf = 3$) > pipelining($pf = 4$) > averaging.

The reason is that pipelining method with small pf may overlap communication time with computation time, while that with large pf does not and requires more communication traffic. In spite of the simplicity of its communication scheme, averaging method has to exchange more information, thus spends more time in communication. However, some amount of time may have been saved because of its asynchronous nature.

(3) BP variations have higher efficiency than the SD variations.

Text to Speech Conversion Test

This experiment is based on the NETtalk experiment of [SR87] in which a neural net is trained using BP to pronounce English text, i.e., to convert a string of letters to a string of phonemes. Through the efforts of T.J. Sejnowski and others, a corpus of 20008 words was created for the NETtalk experiment. A small

subset of 100 words, randomly selected from the Corpus and concatenated together to form a continuous text is used to evaluate the performance of the algorithms. The test procedure is adopted from [SR87]. Each letter is represented *locally* by 27 binary inputs, and *distributed* representation is employed in encoding phonemes by 26 binary outputs.

Test Details

- Number of learning patterns: 828 (100 words with inter-word spacing, giving 834 characters).
- Number of input units: 189 (27 inputs for each letter in the 7-letter window).
- Number of output units: 26 (21 for articulatory features plus 5 for stress and boundary).
- Size of net with 30 hidden units: $30 \times (189+26) = 6450$.
- Weights of the neural network are initialized to random real numbers in the range of -0.3 to 0.3.
- Terminating criterion: 90% correctness in maximum 30 epochs.
- Message size: 52.496*Kbytes* for each of PSBP, PSSD, PPBP and PPSD, and 104.544 *Kbytes* for PABP and PASD.
- Measure of correctness: an output bit is said to be learned if the actual output deviates from the desired one by an amount less that or equal to 0.2.

Comparisons are done on a 'likely' optimal setting under which maximum exact matches are attained with 30 hidden units at the end of 30 epochs.

Test Results and Observations

Optimal results for text-to-speech conversion test appear in Table 9 while the effect of cube size on a net with 30 hidden units is given in Graph 6. Some observations are as follows:

A. With regard to the optimal results of text-to-speech conversion test on a 4-cube:

   (1) Only the BP variations work! All the SD variations reach local minima after few learning epochs.

   (2) All BP variations used almost the same amount of time to complete an epoch.

   (3) All BP variations perform best (in terms of percentage of correctness) with the least number of processors; that is, 1 for PSBP or PABP, and 2 for PPBP.

B. With regard to the effect of hypercube dimension:

   (1) For all BP variations, performance deteriorates with increasing hypercube dimension.

   (2) PABP outperforms PSBP in cubes of dimensions greater than 1.

   (3) PPBP performs better with increasing pipelining factor until it reaches ≈ 13.

C. With regard to the effect of continuous learning:

   (1) For PABP, the net continues to learn as more patterns are presented. This is quite surprisng considering that PSBP in 5-cube does not work at all, while PABP in 5-cube reaches $\approx$60% correctness after 90 training epochs, and PABP in 2-cube passes the 90% correctness mark.

   (2) For PPBP, performance initially increases with more presentations and saturates after $\approx$40 epochs for $pf = 13$, and $\approx$50 epochs for $pf = 6$. Also neither achieve the 90% correctness mark in both cases.

   (3) Although within 30 training epochs most methods applied on large cube does not reach 90% correctness, some of them do give good results in relatively short time.

Table 2: Comparison of characteristics of the two experiments

COMPARISON	classify vowels	convert text-to-speech
Inherent Parallelism	–	+++
Number of Patterns	–	+
Net Size	–	++
Number of Epochs Req'd	++	–
Process Time/Epoch	–	++

(– means less while + means more)

Conclusions

The two tested problems behave very differently under the various parallelized BP methods. The evaluation of the methods is difficult because of the many parameters involved. The vowel recognition has no preference to the approaches SD or BP that the learning schemes adopt. It gives similar results with all of the six parallelized BP methods. On the other hand, SD approach fails in speech-to-text conversion. The conversion seems to favor relaxation, requires more prompt

incorporation of past experience (learned knowledge), and prefers generalization of knowledge combined from a smaller but reasonably rich set of 'dissimilar' patterns learned in parallel.

Of the three parallelized BP methods: standard, pipelining and averaging (SD or BP), pipelining method with an appropriate value of the pipelining factor learns better than the standard method. The value of the pipelining factor should be large enough to incorporate new knowledge to the neural net more promptly but yet small enough so that sufficient patterns are learned in order to derive novel knowledge. One of the drawbacks of the method is that it dedicates one of the processors to manage communication among the rest. The good performance of the averaging method was a big surprise. This method performs much better than the standard method in text-to-speech conversion in cubes of all sizes. Its performance on prolonged training with more pattern presentations is also better than the pipelining method, as the latter soon saturates.

A detailed analysis of the parallel variants of Back Propagation based training algorithms for speech related problems reported here is in preparation [EM90].

References

[BT89] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, 1989.

[Dun88] T. H. Dunigan. Performance of a second generation hypercube. Technical Report ORNL/TM–10881, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Mathematical Sciences Section, November 1988.

[EM90] Ö. Eğecioğlu and B. K. Mak. Performance of back propagation algorithms for speech recognition and synthesis. In preparation, 1990.

[EZ87] J. L. Elman and D. Zipser. Learning the hidden structure of speech. ICS Report 8701, Institute for Cognitive Science, University of California at San Diego, Feb. 1987.

[HEW88] G. D. Haan, Ö. Eğecioğlu , and H. Wakita. Improving the performance of back propagation – trained vowel classifiers. *The Journal of the Acoustical Society of America*, 84, 1988. supplement 1, U4.

[Int88] Intel Corp., Beaverton, Oregon. *iPSC/2 User's Guide*, 1988.

[Lip87] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, April 1987.

[PMT86] S. M. Peeling, R.K. Moore, and M.J. Tomlinson. The multi-layer perceptron as a tool for speech pattern processing research. *Proc. IoA Autumn Conference on Speech and Hearing*, 1986.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1:Foundations. MIT Press, 1986.

[Sch87] Robert Schnabel. An overview of parallel architecture and algorithm characteristics. *SIAM*, Oct 1987.

[SR86] T. J. Sejnowski and C. R. Rosenberg. Nettalk: A parallel network that learns to read aloud. Technical Report JHU/EECS–86/01, John Hopkins University, 1986.

[SR87] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

[SS85] Y. Saad and M. H. Schultz. Data communication in hypercubes. Research Report YALEU/DCS/RR–428, Computer Science Department, Yale University, October 1985.

[SS88] Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Transaction on Computers*, 37:867–872, 1988.

Table 3: Results of echo test (where H = Hamming distance)

LENGTH	COMMUNICATION TIME (ms)				
(*byte*)	H = 1	H = 2	H = 3	H = 4	H = 5
0	0.315	0.325	0.335	0.350	0.360
16	0.385	0.395	0.405	0.415	0.425
32	0.390	0.395	0.410	0.420	0.430
64	0.395	0.400	0.415	0.430	0.430
100	0.400	0.410	0.420	0.435	0.445
128	0.750	0.780	0.810	0.850	0.870
256	0.800	0.820	0.865	0.895	0.925
512	0.885	0.915	0.950	0.985	1.010
1024	1.075	1.100	1.135	1.175	1.195
2048	1.450	1.480	1.505	1.545	1.565
4096	2.175	2.205	2.240	2.270	2.300
8192	3.650	3.675	3.710	3.750	3.770
16384	6.590	6.610	6.655	6.690	6.715

Table 4: Estimates of iPSC/2 communication parameters

HAMMING DISTANCE	STARTUP TIME (ms)	TRANSMISSION RATE ($\mu s/byte$)	COEF. OF CORRELATION
1	0.7065	0.3592	0.999998
2	0.7344	0.3587	0.999997
3	0.7678	0.3593	0.999999
4	0.8041	0.3593	0.999999
5	0.8278	0.3593	0.999999

Table 5: Results of broadcast tests

(a)

LENGTH ($byte$)	1-CUBE BROADCAST TIME (ms)		
	BIBF	HS	SS
16	0.70	0.70	0.70
32	0.70	0.72	0.70
64	0.72	0.72	0.70
100	0.72	0.72	0.72
128	1.10	1.06	1.06
256	1.16	1.12	1.12
512	1.26	1.22	1.20
1024	1.44	1.40	1.38
2048	1.80	1.76	1.76
4096	2.54	2.50	2.50
8192	4.02	3.96	3.96
16384	6.96	6.92	6.90

(b)

LENGTH ($byte$)	2-CUBE BROADCAST TIME (ms)		
	BIBF	HS	SS
16	0.88	1.14	1.20
32	0.88	1.14	1.20
64	0.88	1.14	1.22
100	0.90	1.16	1.22
128	1.64	1.84	2.36
256	1.74	1.96	2.42
512	1.92	2.12	2.58
1024	2.28	2.48	3.10
2048	3.02	3.24	4.16
4096	4.48	4.70	6.32
8192	7.42	7.64	10.76
16384	13.28	13.52	19.66

(c)

LENGTH ($byte$)	3-CUBE BROADCAST TIME (ms)		
	BIBF	HS	SS
16	1.06	1.54	2.18
32	1.06	1.54	2.18
64	1.06	1.56	2.22
100	1.06	1.58	2.24
128	2.16	2.62	4.98
256	2.32	2.76	5.08
512	2.58	3.02	5.34
1024	3.14	3.58	6.54
2048	4.22	4.68	9.04
4096	6.42	6.90	13.98
8192	10.82	11.32	24.38
16384	19.62	20.14	45.16

(d)

LENGTH ($byte$)	4-CUBE BROADCAST TIME (ms)		
	BIBF	HS	SS
16	1.22	1.96	4.12
32	1.22	1.98	4.14
64	1.22	1.98	4.20
100	1.24	2.02	4.24
128	2.68	3.40	10.26
256	2.86	3.60	10.46
512	3.24	3.96	10.92
1024	3.96	4.68	13.50
2048	5.44	6.16	18.94
4096	8.34	9.10	29.54
8192	14.24	15.00	51.70
16384	25.96	26.76	96.28

Table 6: Results of vector addition tests

(a)

LENGTH (*byte*)	VECTOR SIZE	1-CUBE ADDITION TIME (*ms*)		
		ADE	HG	RC
16	4	0.90	0.84	0.82
32	8	1.00	0.92	0.92
64	16	1.22	1.12	1.12
100	25	1.42	1.36	1.36
128	32	2.32	1.88	1.90
256	64	3.16	2.72	2.74
512	128	4.86	4.38	4.40
1024	256	8.30	7.70	7.76
2048	512	15.24	14.36	14.48
4096	1024	29.14	27.68	27.92
8192	2048	56.84	54.30	54.78
16384	4096	112.84	107.56	108.54

(b)

LENGTH (*byte*)	VECTOR SIZE	2-CUBE ADDITION TIME (*ms*)		
		ADE	HG	RC
16	4	1.54	1.42	1.50
32	8	1.72	1.58	1.72
64	16	2.02	1.88	2.12
100	25	2.36	2.22	2.58
128	32	4.00	3.18	4.04
256	64	5.28	4.46	5.68
512	128	7.84	6.98	9.02
1024	256	13.04	12.04	15.74
2048	512	23.70	22.20	29.22
4096	1024	44.70	42.46	56.18
8192	2048	87.04	82.96	109.94
16384	4096	172.22	164.02	217.88

(c)

LENGTH (*byte*)	VECTOR SIZE	3-CUBE ADDITION TIME (*ms*)		
		ADE	HG	RC
16	4	2.22	2.04	2.76
32	8	2.42	2.22	3.16
64	16	2.84	2.64	3.98
100	25	3.32	3.10	4.88
128	32	5.72	4.50	8.42
256	64	7.42	6.22	11.70
512	128	10.84	9.66	18.32
1024	256	17.86	16.50	31.62
2048	512	32.30	30.24	58.36
4096	1024	61.20	57.68	111.90
8192	2048	119.32	112.40	219.32
16384	4096	231.90	222.06	436.58

(d)

LENGTH (*byte*)	VECTOR SIZE	4-CUBE ADDITION TIME (*ms*)		
		ADE	HG	RC
16	4	2.92	2.64	5.18
32	8	3.16	2.90	6.00
64	16	3.68	3.40	7.62
100	25	4.24	3.98	9.42
128	32	7.42	5.82	17.12
256	64	9.54	7.96	23.76
512	128	13.80	12.26	37.04
1024	256	22.66	20.84	63.68
2048	512	40.92	38.10	116.96
4096	1024	77.92	72.50	224.00
8192	2048	152.66	141.10	440.02
16384	4096	303.38	278.50	873.34

Table 6: (e) where HG* time = HG time + broadcast time of vector sum

LENGTH (*byte*)	VECTOR SIZE	5-CUBE ADDITION TIME (*ms*)			
		ADE	HG*	HG	RC
16	4	3.56	4.62	3.24	9.96
32	8	3.90	4.92	3.54	11.60
64	16	4.50	4.54	4.16	14.80
100	25	5.20	6.24	4.84	18.40
128	32	9.12	9.94	7.12	34.66
256	64	11.70	12.78	9.70	48.08
512	128	16.86	18.40	14.88	74.98
1024	256	27.60	29.66	25.20	128.44
2048	512	49.80	52.22	45.92	235.58
4096	1024	94.10	97.24	87.28	450.56
8192	2048	183.50	187.22	169.94	881.58
16384	4096	363.92	366.88	334.98	1747.24

Table 7: Optimal results of vowel classification test on a 4-cube
(pf = pipelining factor, H = number of hidden units,
η = learning rate, N = number of epochs required,
T = processing time, UP = number of patterns not learned)

METHOD	OPTIMAL RESULT					
	H	η	N	$T(min)$	UP	$T/N(ms/epoch)$
PSSD	4	0.2	3568	7.60	21	128
PSBP	6	0.1	3482	13.7	18	236
PSBP†	4	0.1	5522	15.8	19	171
PPSD(pf=2)	4	0.3	4677	10.1	19	130
PPBP(pf=2)	4	0.2	4492	13.2	20	176
PPSD(pf=3)	6	0.2	2708	8.5	27	265
PPSD(pf=3)†	4	0.2	5426	12.0	24	144
PPBP(pf=3)	6	0.2	2213	9.6	22	259
PPBP(pf=3)†	4	0.2	4631	14.8	27	192
PPSD(pf=4)	4	0.5	2318	6.10	26	158
PPBP(pf=4)	4	0.4	1971	6.80	26	207
PASD	4	0.7	4873	12.1	16	148
PABP	4	0.8	9784	31.0	21	190

(† not optimal case but used for comparison)

Table 8: Efficiency results of vowel classification test with optimal setting
(pf = pipelining factor, T_i = processing time for i processors,
E_i = efficiency for i processors, N = number of epochs required)

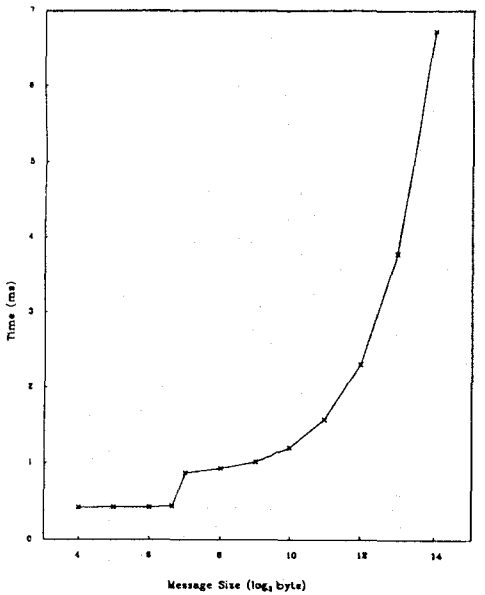
METHOD	RESULT	0-CUBE	1-CUBE	2-CUBE	3-CUBE	4-CUBE	5-CUBE
PSSD	$T_i(min)$	80.4	41.2	21.2	12.6	7.60	4.90
	N	3243	3273	3249	3539	3568	3371
	$T_i/N(ms)$	1488	755	392	214	128	87.2
	E_i	100%	98.4%	95.2%	87.3%	73.4%	53.5%
PSBP	$T_i(min)$	168.1	145.7	56.3	23.7	13.7	8.30
	N	3317	5682	4284	3406	3482	3389
	$T_i/N(ms)$	3041	1539	789	417	236	147
	E_i	100%	98.8%	96.3%	91.1%	80.7%	64.5%
PPSD pf=2	$T_i(min)$		126.0	41.3	22.8	10.1	8.40
	N		5078	4846	5685	4677	5664
	$T_i/N(ms)$		1489	511	241	130	89.0
	E_i		100%	97.1%	88.3%	76.4%	54.0%
PPBP pf=2	$T_i(min)$		169.7		26.5	13.2	10.5
	N		4694		4670	4492	5502
	$T_i/N(ms)$		2169		340	176	115
	E_i		100%	—	91.1%	82.2%	60.8%
PPSD pf=3	$T_i(min)$		323.5	44.5	15.8	8.50	5.50
	N		9438	3770	2793	2708	2392
	$T_i/N(ms)$		2056	708	339	189	138
	E_i		100%	96.8%	86.6%	72.5%	48.1%
PPBP pf=3	$T_i(min)$		118.9	33.1	21.8	9.60	6.60
	N		2339	1910	2688	2213	2233
	$T_i/N(ms)$		3049	1041	487	259	177
	E_i		100%	97.6%	89.4%	78.5%	55.6%
PPSD pf=4	$T_i(min)$		61.7	18.5	9.30	6.10	4.50
	N		2471	2112	2123	2318	2095
	$T_i/N(ms)$		1498	526	263	158	129
	E_i		100%	94.9%	81.4%	63.2%	37.5%
PPBP pf=4	$T_i(min)$		105.5			6.80	6.00
	N		2903			1971	2340
	$T_i/N(ms)$		2181			207	154
	E_i		100%	—	—	70.2%	45.7%
PASD	$T_i(min)$					22.1	41.5
	N					4873	10000
	$T_i/N(ms)$					300	249
	E_i	—	—	—	—	—	—
PABP	$T_i(min)$	51.9	28.9	27.4		60.4	
	N	1434	1564	2661		9784	
	$T_i/N(ms)$	2172	1109	618		370	
	E_i	100%	98.0%	87.8%	—	36.8%	—

(— indicates unsuccessful learning)

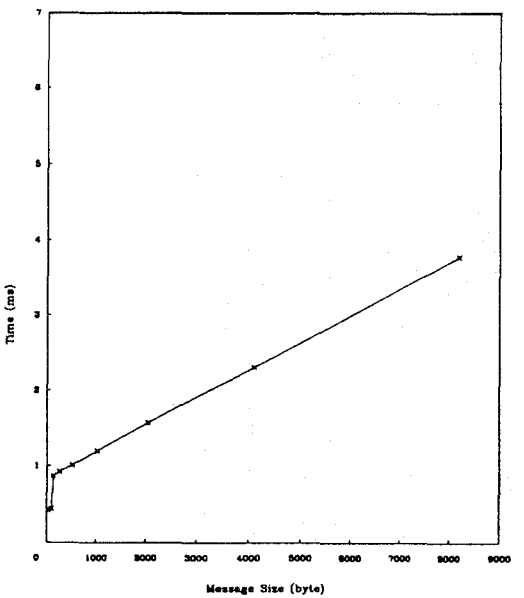
Table 9: Optimal results of text-to-speech conversion test after 30 epochs
(pf = pipelining factor, H = number of hidden units,
P = number of processors required, η = learning rate,
T = processing time, PC = percentage of correctness)

METHOD	OPTIMAL RESULT, H = 30				
	P	η	$T(min)$	$PC(\%)$	$T/30(min/epoch)$
PSBP	1	1.0	161.7	91.4	5.39
PABP	1	1.0	161.7	91.4	5.39
PPBP(pf=1)	2	1.0	161.4	52.9	5.38
PPBP(pf=2)	2	0.9	166.1	62.0	5.54
PPBP(pf=3)	2	1.0	162.2	70.2	5.41
PPBP(pf=4)	2	1.0	162.1	74.6	5.40
PPBP(pf=5)	2	1.0	162.2	79.7	5.41
PPBP(pf=6)	2	1.0	162.5	88.8	5.42
PPBP(pf=7)	2	1.0	162.6	86.7	5.42

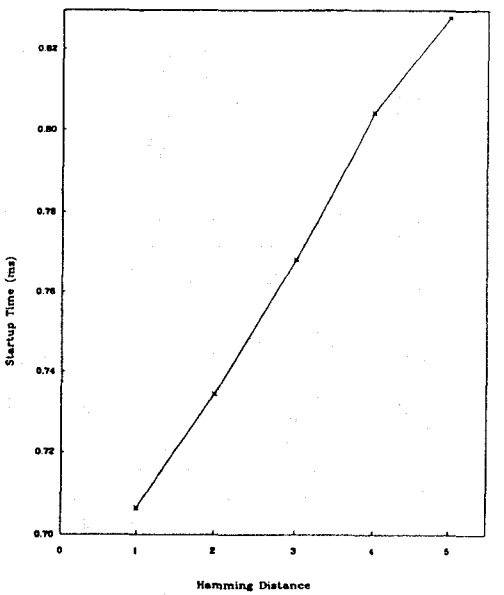
(Note: steepest descent variations do not work.)



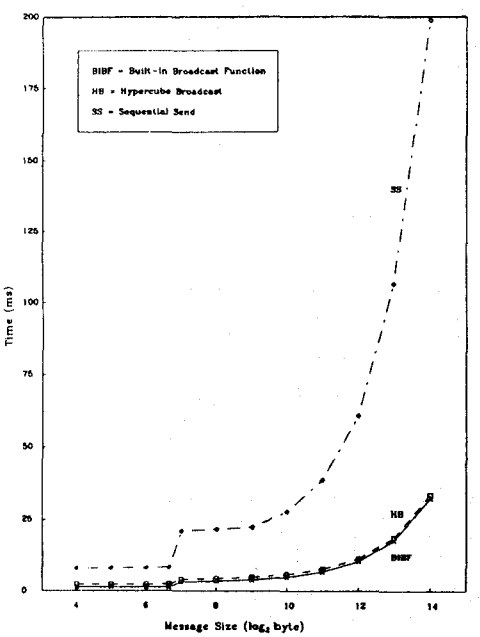
Graph 1: 5-hop communication time (log scale)



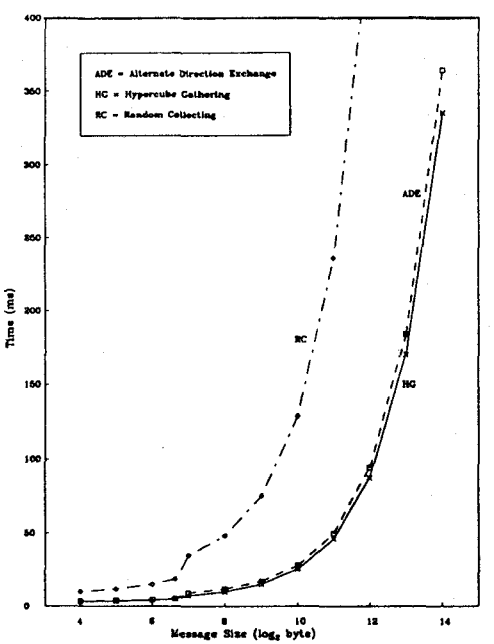
Graph 2: 5-hop communication time (linear scale)



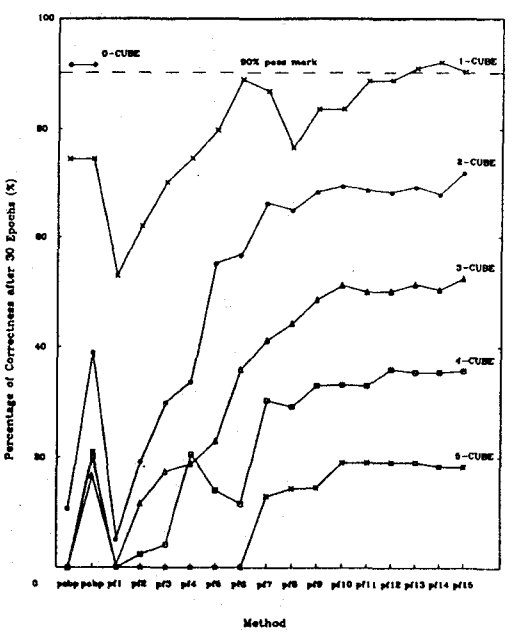
Graph 3: Different hamming distance communication startup time



Graph 4: 5-cube broadcast time



Graph 5: 5-cube vectors addition time



Graph 6: Effect of cube size on text-to-speech conversion with H=30