JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 6, 29 1-330 (1989)

Mapping Neural Networks onto Message-Passing Multicomputers*

JOYDEEP GHOSH† AND KAI HWANG

Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089-0781

Received May 1, 1988

This paper investigates the architectural requirements for simulating neural networks using massively parallel multiprocessors. First, we model the connectivity patterns in large neural networks. A distributed processor/memory organization is developed for efficiently simulating asynchronous, value-passing connectionist models. On the basis of the network connectivity and mapping policy, we estimate the volume of messages that need to be exchanged among physical processors for simulating the weighted connections of a neural network. This helps determine the interprocessor communication bandwidth required, and the optimal number and granularity of processors needed to meet a particular cost/performance goal. The suitability of existing computers is assessed in the light of estimated architectural demands. The structural model offers an efficient methodology for mapping virtual neural networks onto a real parallel computer. It makes possible the execution of large-scale neural networks on a moderately sized multiprocessor. These mapping techniques are useful to both architects of new-generation computers and researchers on neural networks and their applications. Until the technology for direct hardware implementation of large neural networks is available, simulation remains the most viable alternative for the connection& community.

1. ARE EXISTING COMPUTERS ADEQUATE FOR NEURAL SIMULATION?

Connectionist models of computation [16] have been in the limelight recently as promising alternatives to traditional approaches to solving complex problems in artificial intelligence [14, 46]. In particular, they have been advocated for inferencing systems with learning capabilities, pattern recognition, computer vision, and robot control [19, 38].

* This research was supported in part by the NSF under Grant DMC-84-2 1022 and in part by the ONR under Contract N00014-86-K-0559.

† Currently with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712-1084.

0743-7315/89 $3.00

A salient feature of connectionist models is that they involve a large number of elementary computations that can be performed in parallel [15]. Memory is completely distributed, processing is asynchronous, and the models are inherently tolerant to malfunctioning processing units or connections [28]. Consequently, they are amenable to a highly concurrent hardware implementation using thousands of simple processors, provided adequate support is available for inter-processor communication.

This paper examines the architecture of parallel computers from the viewpoint of their ability to effectively simulate large ***neural networks.*** The term "neural network" is used ***in a broad sense*** to cover artificial neural systems and other connectionist models, instead of having a restricted biological connotation. Our approach is summarized in Fig. 1, which outlines the organization of this paper. We do not focus on knowledge representation and learning in a neural network. Instead, we explore ways of effectively implementing neural networks on a highly parallel computer.

At present, a number of research groups are using commercially available parallel computers for neural network simulations. These include coarse/

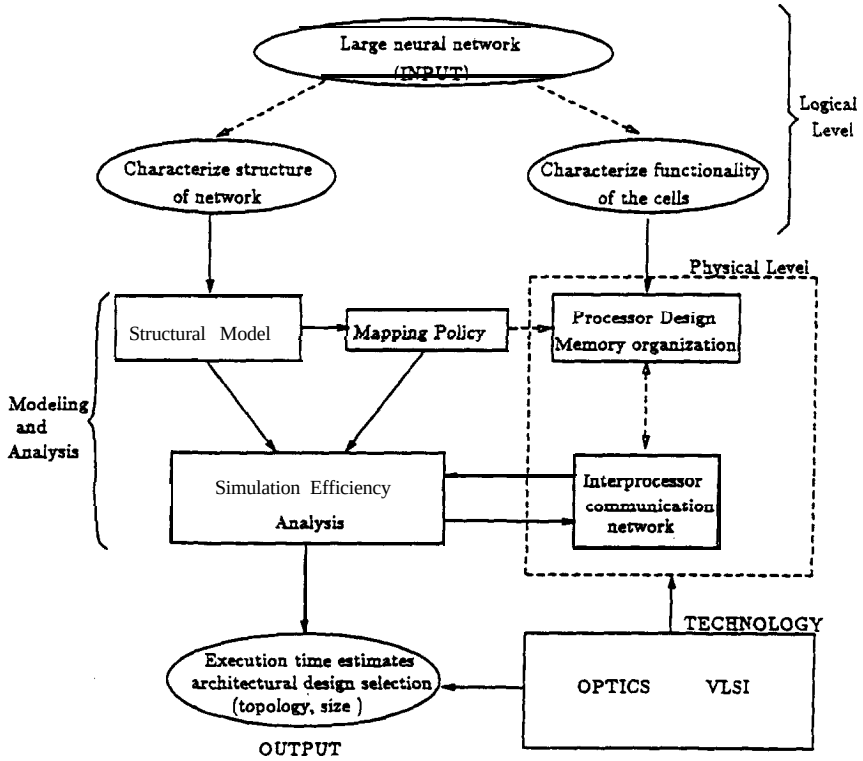

FIG. 1. Critical issues in developing a multicomputer for neural network simulation.

medium-grained systems such as the transputer-based Computing Surface with 42 processors at Edinburgh [18], the Warp [43], and the 128-node BBN Butterfly at Rochester [17]. In addition, massively parallel computers, such as the AMT DAP [18] and the Connection Machine [20], are being used.

Are any of these systems adequate for simulating very large neural nets? To answer this key question, we use a structural characterization of the application domain to predict architectural demands. The results indicate that indeed, none of the currently available machines can efficiently simulate a neural network with $O(10^8)$ interconnects at a speed of 10^9 interconnects/second, which corresponds to the computational capabilities of a fly [12]. Most coarse/medium-grained systems are not able to provide sufficient interprocessor communication bandwidth. Moreover, their powerful processors are an overkill, since computational requirements are largely confined to calculating matrix-vector products, which form the "inner loop" for neural systems. On the other hand, massively parallel, fine-grained systems such as the Connection Machine are seen to have insufficient local memory. This forces the use of secondary storage, which causes the I/O transfer rate to be the most critical bottleneck. These inadequacies provide the motivation for developing tailor-made computer architectures to cater to such large-scale simulations in the long run.

The rest of the paper is organized as follows: In Section 2, we present a model to capture the essential structural features of large neural networks using a few parameters. This model provides a way of mapping these networks onto multicomputer topologies. We also characterize the functionality of a cell in asynchronous, value-passing computational models. This leads to the distributed processor/memory organization presented in Section 3. This architecture is tailored to the efficient simulation of asynchronous, value-passing neural networks.

A central theme of this paper is analyzing *interprocessor communication bandwidth requirements* during simulation, since this is perceived to be a major bottleneck during highly concurrent execution. This is done both theoretically and experimentally in Section 4 using the structural model and mapping policy developed in earlier sections. These estimates serve to indicate how long a given neural network will take to execute on a multicomputer. They also provide a metric for determining the suitability of different processor interconnection schemes and for choosing the system size. We conclude this paper by reexamining the suitability of available computer systems for neural network simulations.

2. A Mapping Model of Neural Networks

A connectionist model is distinguished by the use of interconnections among a large number of elementary computing units or cells as the principal

means of storing information, and by the ability to simultaneously operate on this information in a highly parallel and distributed fashion. This viewpoint characterizes most artificial neural systems as well as massively parallel marker/value passing networks and constraint-satisfaction networks [14, 16,461. These systems have some commonalities with the neural networks of the human neocortex, where knowledge is stored and processed by a large number of interconnected neurons. Neurons have elementary computational capabilities, but they operate in a highly parallel fashion. The concurrent processing by aggregates of neurons in a cooperative and competitive manner is fundamental to the functioning of the brain.

Each cell of a connectionist system is capable of performing basic arithmetic or logic operations and has little internal memory but a large number of connections to other cells. Every connection has a *strength* or a *weight* associated with it, and the pattern of weights represents the long-term knowledge of the system, The power of a neural system comes from its ability to simultaneously apply the entire knowledge base to the problem at hand. All the cells operate concurrently and computations are directly affected by knowledge encoded in the network connections. By contrast, a symbolic machine can bring to bear only those representations which can be retrieved from memory.

We view a connectionist network as a component of a cognitive system (Fig. 2) rather than as a stand-alone computer. A similar viewpoint has been expressed by researchers in the Cognitive Architecture Project [28]. The neural network has two sources of inputs:

(i) Raw sensory data, preprocessed through filters, systolic arrays and/or secondary neural networks, and

(ii) Feedback, training and control signals from the user or the inference engine.

The inference engine is built on top of the neural network. It uses more conventional symbolic processing to make decisions based on the output of the neural network and on its own store of high-level knowledge (information, rules, etc.). This inference engine may run on the host computer or on a dedicated machine. For interactive systems, the host or network may also control actuators that interact directly with the environment.

2.1. Logical Structure of Neural Networks

The logical structure of a neural network is given by its underlying network graph that shows the connectivity pattern of weights. Each vertex of this graph represents a cell, and a directed edge exists between two vertices if and only if there is a nonzero weight associated with the corresponding pair of cells. The *connectivity* of a set of cells is the ratio of the number of actual

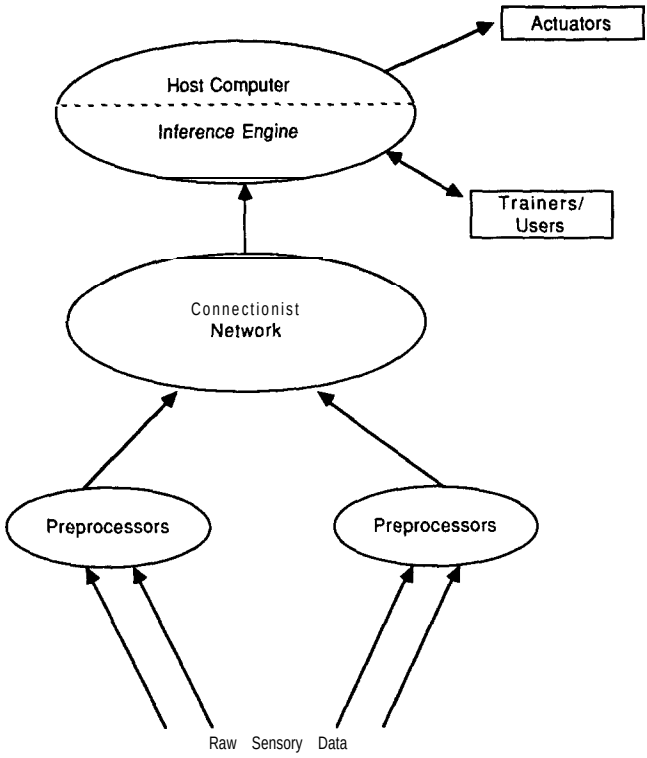

FIG. 2. Architecture of a cognitive system.

connections among these cells to the number of connections if the cells were fully connected.

Table I outlines the structural characteristics of some common network classes. Several common features become apparent in large networks. Fully connected neural networks with symmetric weights can be made to converge to one of a finite set of equilibrium points that are the local minima of the Liapunov or "energy" function [10,321. However, for networks with thousands of cells, full connectivity is computationally prohibitive. This is because the total amount of computation required is directly proportional to the number of interconnects and thus increases quadratically with respect to the system size.

We will focus on the case where it is possible to partition the network into groups of cells such that the connectivity within a group is much higher than the overall network connectivity. Typically the cells of such groups are used to effect the same function or related functions. For example, a winner-take-all network (WTA) [16], used for selecting one out of a set of alternatives, consists of a group of cells that are fully interconnected (conceptually)

TABLE I
STRUCTURAL CHARACTERISTICSOF VARIOUS CONNECTIONIST NETWORKS

Network type	Connectivity	Special features
Crossbar associative networks [32, 39]	Full	For large networks, a sizable number of connections may have zero weights
Semantic networks (local representation) [50]	Sparse	Hierarchical organization
Semantic networks (distributed representation) [52]	Moderate	High connectivity within clusters; few connections among clusters
Multilayered networks [42, 48]	Moderate	Most connections are between adjacent layers or within a layer
Human neocortex [13]	Moderate to high	Hierarchical organization; spatial locality of connections

through inhibitory links. Another commonly occurring configuration is that of a group of cells which form a mutually supportive coalition by having a number of reinforcing connections among themselves.

The connections among highly connected groups are not random. For example, groups of cells representing competing hypotheses will have interconnects representing inhibitory influences among them but few connections with groups signifying independent hypotheses. The notions of ***competing hypotheses*** and ***stable coalitions*** are a recurrent theme in neural networks that are functionally characterized by the cooperative and competitive behavior of an ensemble of neurons [7]. Competing coalitions of units ***at various levels of abstraction*** form the basis of several neural models such as those for computer vision [8].

Both analytical [23] evidence and neurological [13] evidence suggest that a comprehensive neural network for cognition will have a hierarchical organization. At the lower levels, primary features are extracted from input data with a high degree of concurrency and redundancy. Information is represented and processed in a distributed fashion. At higher levels, inferences are made on the features extracted from lower levels. Representation is more local, and a single concept or attribute is embodied in a small number of cells. Such a model concurs with our view of a cognitive system as reflected in Fig. 2.

2.2. *A Structural Model for Mapping Neural Networks*

In this section, we present a general model to characterize the structure of large neural networks. The model should be able to capture the essential structural features of these networks in a few parameters. Second, the model

should indicate how these networks can be partitioned for efficient mapping onto multiprocessor / multicomputer systems. Finally, a reasonable prediction of interprocessor communication requirements should be obtained on using the mapping scheme for various multicomputer topologies.

To characterize a network, we first partition it into disjoint ***core*** regions of comparable sizes. A core consists of a group of cells that are relatively highly interconnected among themselves. For example, all the cells of a simple WTA network can form a fully connected core. Though the core need not be fully connected, the probability of finding a direct connection between two cells should be much higher if those cells belong to the same core. Of course, it may be possible to have a subgroup of cells within a core that is even more highly connected.

It is desirable to partition the network into groups almost equal in size, with relatively few interconnections among groups. A practical way of specifying a very large network is through a hierarchical procedure where **subnetworks** are distinguished by their functional properties and are gradually specified at finer levels of detail [28]. Partitioning schemes which follow from this hierarchical decomposition usually give good results. This is because cells achieving a common function or goal tend to have more interrelationships among themselves than with cells belonging to subnetworks for disparate functions [17]. For more randomly structured networks, simple techniques developed for partitioning of graphs [37] prove effective. The idea here is not to spend a lot of computation in trying to find optimum partitions, but to come up with a reasonable one.

Cores whose constituent cells have a substantial number of intercore connections are further grouped into ***influence regions.*** For instance, consider a core consisting of cells in a mutually supportive coalition. The cores representing competing coalitions would be natural choices for inclusion in the influence region of this core. The influence region of a cell is identical to the influence region of the core to which the cell belongs.

For simplicity, we require the symmetric and transitive properties to hold, so that, for any three core regions, A, ***B,*** and C,

(1) ***B*** is in the influence region of A, if and only if A is in the influence region of ***B***;

(2) If ***B*** and C are in the influence region of A, then they are in the influence region of each other.

Then the entire network can be partitioned into disjoint sets of cells such that each set is precisely the union of the core and the influence region of an arbitrary cell in that set. Moreover each cell can now distinguish among three groups of cells, namely, those in its core, those in its influence region, and the rest of the cells forming the ***remote*** region for this cell, as illustrated in Fig. 3a.

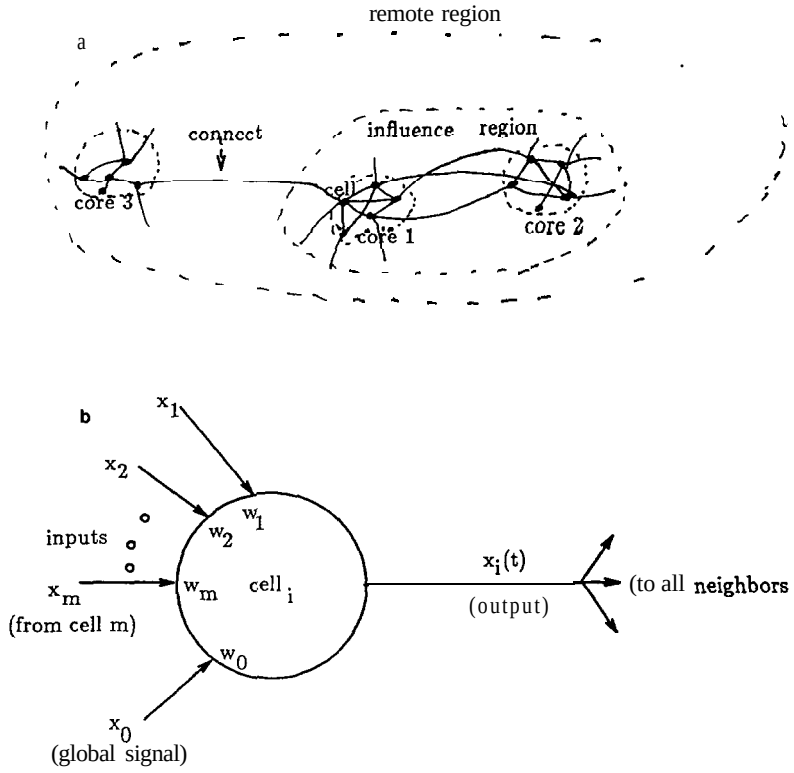

FIG. 3. (a) The universe of a processing cell. (b) A generic processing cell of a neural network.

As an illustration, consider the celebrated approach to the traveling salesman problem (TSP) based on the Hopfield model [33]. An $N = n \times n$ matrix of cells is employed to solve an n-city tour problem. Each cell is connected to all other cells in the same row or column, as well as to all cells in the two columns immediately preceding or succeeding its own. A cell need not be connected to any other cell besides the $4n - 3$ cells specified above. Note that the corresponding network graph has degree $0(\sqrt{N})$, even though the computational model of Hopfield assumes full connectivity.

For this TSP network, suppose we define the core associated with a cell to be composed of all the cells in its column. The network then has n equal-sized cores, each of which is fully connected internally. Furthermore, let the influence regions be formed from cells in four consecutive columns. Thus the influence region of a cell in column 1 comprises all cells in columns 2, 3, and 4; the influence region of a cell in column 6 is precisely the cells of columns 5, 7, and 8; and so on. This partitioning is consistent with the requirements of symmetry and transitivity. For $n = 2048$, the probability that a

connection exists between a cell and another one in its influence region is about 0.5, while the probability that it is connected to one of the remaining cells is less than 0.0005. In general, the probability that it has a connection to another cell decreases as we go from the core region to the influence region and finally to the remote region.

Let G_i be the number of cells in the influence region of cell a, and C_i be number of connections of a to this region. Let the probability that a is connected to a cell b in its influence region be denoted as $P_i(a - b)$. If the connections to the influence region were randomly distributed, then $P_i(a - b) = C_i/G_i$. However, in large neural networks these connections often tend to be concentrated toward a few areas rather than being random. In such cases, the probability that a cell a is connected to a cell b increases if there is another cell c in the core of b that is connected to both a and b. We denote this ***conditional probability*** as pi. Thus $\rho_i = P_i(a - b$ given $a -$ c and $b -$ c). Random connections to the region of influence are modeled by $\rho_i = C_i/G_i$. Higher values of ρ_i denote a more clustered connectivity pattern for this region. Similarly, we define $\rho_r = P_r($ a $-$ b given a $-$ c and $b -$ c) when b and c are in the remote region of a.

We characterize the structure of a large neural network (CN) by an **eight-tuple,**

$$CN = \langle M, G_c, G_i, C_c, C_i, C_r, \rho_i, \rho_r \rangle,$$

where Mis the total number of cells in the network; G_c and G_i are the average number of cells in the core and influence regions, respectively; and $C_c, C_i,$ and C_r represent the average number of connections from a cell to another in the core, influence region, and remote region, respectively. Thus the average number of connections per cell is $C_c + C_i + C_r$. The conditional probabilities, ρ_i and ρ_r, indicate the amount of clustering or lack of randomness in the connection patterns of the network. A network with completely random interconnects has very low values for ρ_i and ρ_r, while values near one are typical of highly localized interconnects. In Section 4, we show that these two parameters have a dramatic effect on the communications support required to simulate the network.

Table IV in Section 4 contains five sets of parameters which characterize the networks which are considered for performance analysis in that section. Note that all parameters in the eight-tuple are *mean* values of the actual distribution of these parameters over all cells. We assume that the variance of these distributions is small. This is reasonable if the cores are of comparable sizes, and the regions of influence are chosen appropriately.

A particular type of connectionist network can be structurally characterized in greater detail using a more specific model. A structural description given by this elaborate model can be translated to the general model. This has been illustrated for multilayered networks in [24].

2.3. Functional Behavior of Individual Cells

To determine the processing and memory requirements of a virtual connection&t machine, we use the following characterization of the behavior of each cell in a value-passing neural network: The generic cell, shown in Fig. 3b, receives inputs that are either global signals or the output signals of neighboring cells modulated by the weight of the connection between them. At any time, cell i has an *internal state*, $q_i(t)$. The number of possible internal states is small, usually 10 or less [16]. Each cell also has an output $x_i(t)$ denoting its activation level. The output assumes integer values from a small range, say $(-127$ to $128)$, and can thus be represented by 1 byte. The *state* of a cell comprises its internal state and output. Each cell continuously updates its state and passes its new output value to all neighboring cells.

The state of a cell is updated according to the equations

$$x_i(t+1) = f_{q_i(t)}[\sum w_{i,j}(t)x_j(t)] \tag{1}$$

$$q_i(t+1) = g[q_i(t), x_i(t), x_0(t)]. \tag{2}$$

$x_j(t)$ is the output of cell j (as seen by cell i) at time t, and $w_{i,j}(t)$ is the weight of the interconnect between cells i and j. By letting $w_{i,i} \neq 0$, we also cater to autofeedback. The *activation function*, f, may depend on the current internal state, which signifies, for example, whether the cell is disabled, saturated, or in normal operation. Typically, f is a nonlinear threshold function. Note that x_0 is a special dummy cell used to cater to global signals which can be used for broadcasting the thresholds and enable/disable and begin/end signals.

In addition to the ability to update their states, the cells of an adaptive system have the ability to modify the weights $w_{i,j}$ governing their behavior. Autonomous learning mechanisms [7] typically use a variation of Hebb's law where the strength of a connection is changed by an increment proportional to the product of the activation levels of the two connecting cells. Supervised networks [46], on the other hand, usually employ a gradient descend algorithm to minimize an error function. For example, each weight may be changed to reduce a local error given by the difference between the present output $x_i(t)$ and a desired output x:(t). In both cases, we note that the weights can be updated locally.

3. Multicomputers for Neural Network Simulation

An artificial neural system can be either fully implemented in hardware or virtually implemented in software [29]. A full implementation provides a dedicated processor for each cell, and a dedicated information channel for

each interconnect. Modest-sized fully implemented analog systems have been realized on VLSI chips [34]. Electrooptical implementations are also being explored [5 1]. Once built, fully implemented systems are difficult to reconfigure. However, they are attractive for certain special-purpose applications such as sensory processing and low-level vision analysis [29].

Virtual implementations simulate the function of a neural network using fewer physical processors, by time multiplexing several cells on each processor. The simulation of a neural network of M cells on a computer system with N processors is a virtual implementation if $N < M$. In this case, the network is partitioned into N disjoint groups of cells. These groups are then mapped onto a multicomputer as illustrated in Fig. 4. The activity of all cells in a group is simulated by a single processor. Interconnects among these groups are multiplexed over the physical interprocessor links. The states of the cells, connection weights, and other system parameters are stored in local memories. Since the implementation is essentially done in software, it is more flexible for general-purpose applications.

Currently, all large neural systems are being virtually simulated [17, 29]. For example, "neurocomputers" that are being marketed for implementing artificial neural systems are essentially conventional personal computers with attached coprocessors or array-processors, and specialized software. Small bus-based multiprocessors such as the TRW Mark III take advantage of the much faster speeds of electronic systems as compared to biological

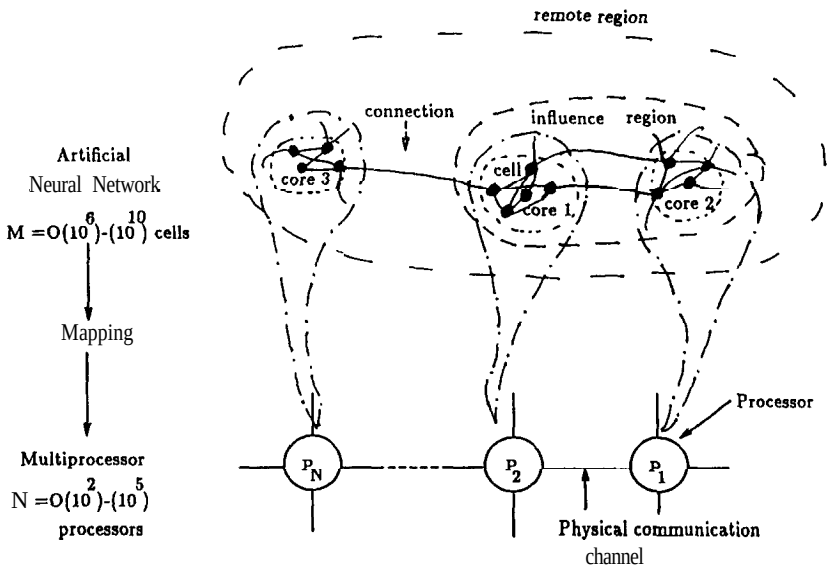

FIG. 4. Mapping of neural nets onto parallel computers.

TABLE II
NEURAL NETWORK SIMULATION HARDWARE

Name, company [Ref.]	Processor type, host	IPS[a]	Interconnects	Remarks"
Mark III, TRW [5]	Eight 68020/68881's on VME bus; VAX/VMS host	500K	1.13M	Runs most neural paradigms
Σ-1, SAIC [4]	Pipelined Harvard architecture; PC/AT	10M	1M	BP, ART, HOP, Learn Matrix
ANZA Plus, HNC [I]	Four stage pipelined Harvard architecture using Weitek XL chips; Zenith Z-386 (Sun 3 for VME)	6M	1.5M	BP, NEO, CP, Avalanche etc.
NX- 16, Human Devices [54]	16 RISC processors; Mac II host	8M	1.6M	Supports McBrain
NDS, Nestor [54]	Software on Sun/Apollo workstation or PC/AT	500K	15M	Pattern recognition (proprietary)
ODYSSEY, TI [54]	DSP accelerators; TI Explorer	20M	256K	General-purpose neural simulator
WARP, CMU [12,431	Linear systolic array of 10 processors	10–17M	320K	BP (NETtalk)
Butterfly, BBN [2, 12]	128 68,020-nodes connected by butterfly network; VAX	16M	120M	Runs Rochester connectionist simulator [17]
CM-2, Thinking Machines Corp. [3, 12]	64K single-bit processors, hypercube network; VAX or Symbolics	13M	64M	BP [9] (NETtalk)
MX-1/16, MIT Lincoln Labs [12, 26]	16 digital signal processors; Lisp machine	120M	50M	Projected performance
X/MP 2, Cray Research [12]	Two Cray processors with shared memory	50M	2M	Estimated performance

Sources. DARPA Neural Network Report (July 1988 [12]), Toborg and Hwang (Jan. 1989 [51]), Wah (Aug. 1988 [54]); literature from TRW [5], SAIC [4], and HNC [I].

[a] IPS = Interconnects per second; BP = Back Propagation; CP = Counter Propagation; NEO = Neocognitron; HOP = Hopfield; ART = Adaptive Resonance.

systems in order to cope with the massive parallelism of the latter. The first six entries of Table II summarize the specifications of leading commercially available neural simulation packages, while the last five entries indicate the computational capabilities of parallel computers that have been considered for neural network simulation.

3.1. Processor and Memory Requirements

We are interested in the implementation of neural systems with possibly millions of cells, on fine-grained multicomputer systems whose architectures are tailored to meet the requirements of efficiently supporting artificial neural systems and other value-passing connectionist models. Such computers need adequate software support for specifying, editing, and archiving the network, and for interpretation and presentation of the results. Moreover, a comprehensive interactive user interface is indispensable for these systems. In this paper, we shall not explore these software design issues, but confine ourselves to hardware design considerations.

Figure 5a shows the design of a virtual connectionist multicomputer. Figure 5b shows the detailed organization of each processor. A loosely coupled system has been chosen instead of a tightly coupled one with shared memory. This is because data can be partitioned into many segments without much data sharing and dependence, and all cells can be processed concurrently. The internal state of a cell as well as a list of its connections and their weights is accessed only by the processor simulating that cell. However, its output value needs to be conveyed to all its neighbors. These outputs, together with global signals or conditions, are the only source of data sharing.

More importantly, it is not essential that the new output value of a cell be immediately available to its neighbors, since neural computation is quite robust. An exception occurs in applications such as speech processing, where the temporal characteristics of the network are important and message latency can seriously impair the accuracy of simulation. Effective consistency can be ensured if the interconnection network has adequate bandwidth. Therefore, a loosely coupled system is deemed more suitable for neural-network simulation.

The entire network of M cells is partitioned into N disjoint sets of cells of approximately equal sizes, one for each processor. We define the ***homegroup*** of a processor to be the set of cells that are mapped onto it. This processor is ***the home processor*** for all cells of its home group. A processor is a ***virtual neighbor*** of a cell if its home group contains some neighbor of this cell. Two processors are virtual neighbors if one of them is a virtual neighbor of a cell in the other's home group.

Computing with a neural model involves a "continuous" updating of the states of all cells. Some global functions on the cell states are also evaluated

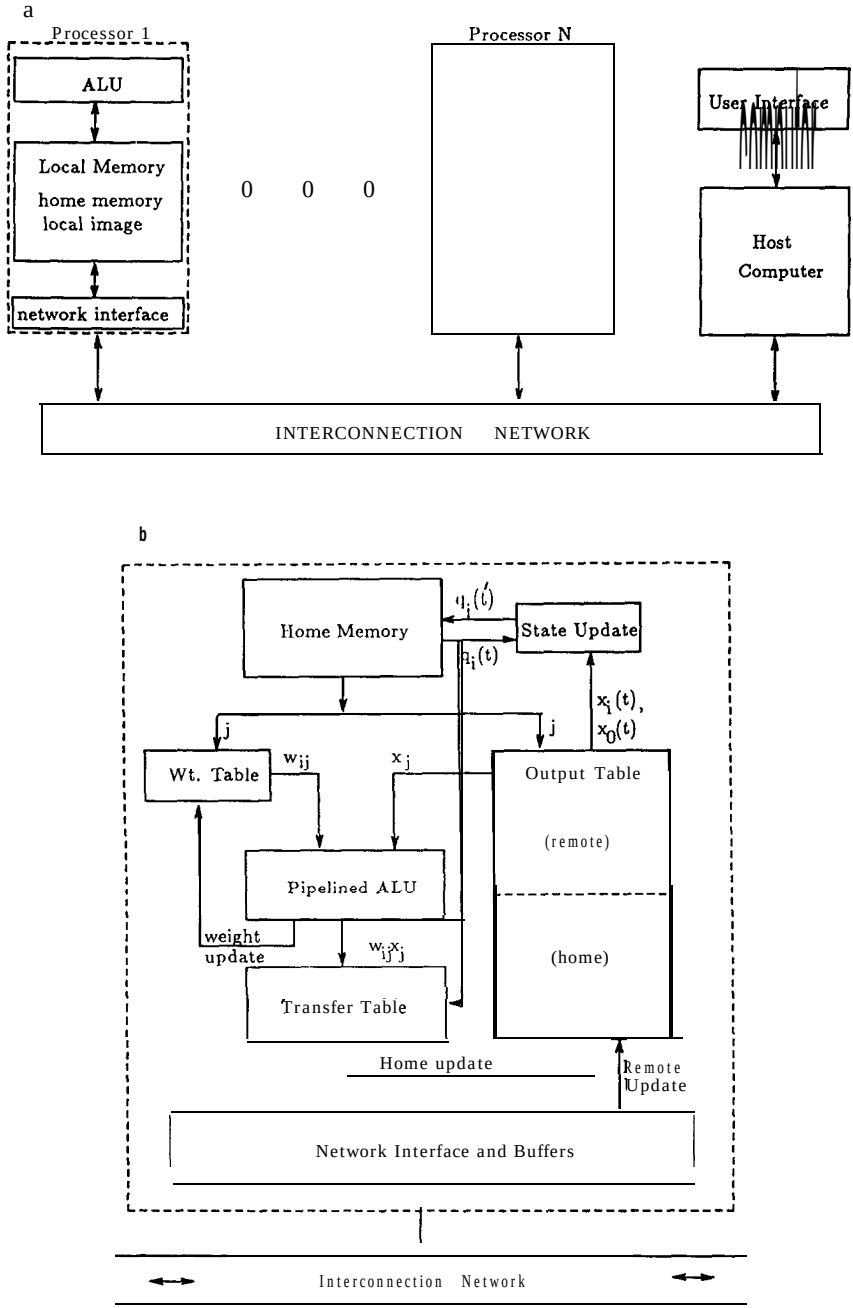

FIG. 5. A multicomputer architecture for simulating neural networks. (a) Overall organization. (b) Architecture of a single processor.

at regular intervals for determining convergence and system outputs. In this paper, a discrete-step version is assumed where the computation is carried out through a sequence of *iteration cycles.* In each cycle, the processors operate concurrently and update the states of their respective home groups. The processors have local clocks, but a macrolevel synchronization is done whereby a processor starts with the next iteration only when all other processors have completed the previous one. Macrosynchronization alleviates the problem of distributed synchronization among a large number of neurons without severely impairing the accuracy of computation when temporal characteristics of the network are important.

The virtual time of an iteration cycle corresponds to the mean value of the time difference between successive state updates of a cell. For each processor, an iteration cycle involves about *M/N unit update* cycles (approximately one for each cell of the home group), and overhead for synchronization and computation of global functions.

Figure 5b shows the organization within each processor. The scheme is similar to that proposed in [29] and to the node architecture of the TRW Mark III [40]. It differs primarily in the network interface which is now tuned for a message-passing environment rather than for a bus-based multiprocessor, and in the implementation of the state update and transfer functions.

The home memory stores the current internal state and a list of neighbors for each cell of the home group. The weight table contains the weights of all connections involving cells of the home group. The output table has two sections: The home section records the current output of the home group. The remote section contains a *local image* of the rest of the network. This image consists of the outputs of those cells that are not in the home group but have a connection with some cell within that group. Global signals that might influence the local computation are also stored. Since communication among the processors is not instantaneous, the local images may not be consistent across the system. However, the local image should reflect outputs that are either current or a few iterations old.

To update the state of a cell, its list of neighbors is read sequentially. This list provides the addresses of successive ($w_{i,j}$, x_j) pairs which are fetched from the weight table and output table, respectively. The processing unit has a pipelined architecture which accumulates the sum of successive products, $w_{i,j}x_j$. For adaptive systems, the processing unit also updates the weight coefficients. The proposed architecture can be extended to incorporate parallel processing within a chip by having several pipelines and multiple data paths so that the states of several cells in the home memory can be updated simultaneously.

The transfer function, T, is implemented by table lookup. If the 8 most significant bits of the weighted sum are used to select one of 256 possible

outputs, a 1-Kbyte RAM suffices for a table lookup for up to four different transfer functions. The internal state of the cell is updated using a finite-state machine. This is also implemented using a table lookup since a cell can assume only a small number of internal states.

The new output of a cell updates the home section of the output table. It is also sent to the network interface, which conveys this information to the cell's virtual neighbors via the interprocessor communication network. This interface also updates the remote section of the output table when it receives new output values from other processors. Each processor also has a communication link to the host for transmitting global signals, initialization, diagnostics, and readouts of system state.

3.2. Choice of Interconnection Network

The value of a cell is passed to a virtual neighbor through a *message.* Due to their small size, it is impractical to send such value-based messages individually. The overheads involved in message setup, destination decode, store, and forward would easily dominate the actual transmission time. We therefore combine values from the same home group that are destined for the same processor into message *packets.* Each packet contains a set of {(internal) cell identity number, output value} pairs, besides header information containing the addresses of the source and destination processors and control bits. The size of a packet which conveys 16 values will be about 50 bytes.

What are the basic considerations in choosing an appropriate interconnection network for a connectionist machine with over 1000 processors? Ideally, the interconnection network should have adequate bandwidth to pass the new output of each cell to all its virtual neighbors within one iteration cycle time. Let B_T be the bandwidth required of the interconnection network to achieve this goal. Due to the small size of the packets and the large number of virtual neighbors per processor, packet switching with *asynchronous communication* and distributed control of message routing is preferred. This means that some packets may not be able to reach their final destination within an iteration time because of contention delays. Experiments have shown that, for moderate loads, the bandwidth of a large system is reduced by about 50% due to contention or unused wires [30]. Thus, if a contention-free bandwidth of $2\ B_T$ is provided, then most of the packets will reach their destination within an iteration cycle. This is acceptable because the computational model is robust enough to prevent a few stragglers from affecting the results. This also indicates that system throughput is more critical than metrics like worst case communication delays.

Multiprocessor systems can be based on four types of networks: direct interconnection networks with a point-to-point (static) topology, multistage

interconnection networks (MINs), crossbars which have a dynamic topology, and bus-connected systems. In MINs, the time to communicate between any two processors increases logarithmically with the system size, even in the absence of resource contention. Since MINs do not provide quick communication between particular processors, they do not benefit from the locality of messages exhibited by the neural networks being considered.

Crossbar switches can be used to provide a pseudo-complete connectivity among the processors. Currently, crossbar switches have fewer than 100 inputs. Recent research in the design of large crosspoint switches using multiple one-sided chips [53] indicates that larger crossbar switches may soon be feasible. Still, the full connectivity offered by crossbars is superfluous, since a good mapping strategy ensures that each processor communicates only with a fraction of the other processors.

Bus-based systems entail more complex hardware for bus allocation and arbitration. A single bus cannot support a large number of processors. However, multiple-bus systems or a system using both buses and direct interconnects can be viable for a medium-sized system, particularly if a lot of broadcasting or partial broadcasting is involved. This issue is further investigated in Section 4.2.5.

Point-to-point topologies are deemed suitable for systems with thousands of nodes due to direct communication paths among the computer modules, simpler communication protocols, and more efficient support for local communication [6]. In particular, the recently proposed techniques of virtual cut-through and wormhole routing significantly reduce the packet-forwarding overhead for such networks [45]. Massively parallel systems based on the hypercube, mesh, or binary tree topologies [35] have already been constructed. A class of modular networks called *hypernets* has been proposed recently for hierarchically constructing massively parallel multicomputer systems [36]. They are constructed incrementally with identical building blocks that are well suited for VLSI implementation. Hypernets provide cost-effective support for a mixture of local and global communication. Like the cube-connected cycles [44], hypernets exhibit a constant node degree and $O(\log N)$ network diameter. However, hypernets are more suitable for incremental construction, since links once formed need not be changed as the system grows in size.

Tree-based topologies are notorious for a communication bottleneck toward the root, though this can be alleviated by augmenting them with extra communication channels [27], or by ensuring that there is little communication between the nodes of the left and right subtrees. Due to their extremely sparse connectivity, these topologies are not being considered for constructing connectionist machines. Large interconnection networks based on hypercubes, hypernets, and the *torus* [49] have attractive properties for implementation on VLSI chips. Efficient algorithms for distributed routing and

broadcasting also exist for them [31, 36]. Therefore, these three topologies are assessed in Section 4 regarding their suitability for supporting a neural network.

3.3. Mapping of Neural Networks

How should we partition the cells of a neural network into home groups? What should the criteria be for allocating these groups to processors? In a fully connected network, these choices are not critical, since all equal-sized partitions are topologically equivalent. But in a partially connected network, proper partitioning and allocation can significantly affect the volume and characteristics of interprocessor communications, and hence the demands on the processor interconnection network.

Let $v_{k,l}$ be the number of cells in the home group of processor k that have processor 1 as their virtual neighbor. Let the distance between two processors be given by the shortest number of communication links traversed from one to the other. Suppose each pair ofprocessors had a dedicated communication link, i.e., they were at a distance of 1. Then, an optimal partitioning of the cells into home groups to minimize the total interprocessor communication bandwidth would be that for which $\sum_k \sum_l v_{k,l}$, $1 \leq k$, $1 \leq N$, is minimum. This would also be the objective function if a crossbar or a single bus was used to provide pseudo-complete connectivity among all processors. For a direct interconnection network, the distance between processors k and 1, $d_{k,l}$, is not constant for all processor pairs. In this case, the partitioning and allocation policies are interrelated and depend both on the structure of the neural network and on the topology onto which it is mapped. The joint optimizing function is

$$\min\left(\sum_k \sum_l d_{k,l} v_{k,l}\right), \qquad 1 \leq k, \qquad l \leq N. \tag{3}$$

Finding the optimal partitioning and allocation is an NP-complete problem. In fact, for a network specified by the general model of Section 2.2, an optimal solution is not possible because the overall network structure is given, but the individual connections are not enumerated. However, the model suggests the following heuristic approach:

Partitioning Principle. Prefer cells belonging to the same core for inclusion in the same home group. If a core is larger than a home group, then this policy tends to minimize the number of home groups to which the cells of a core are distributed. Otherwise, it tends to minimize the number of cores mapped onto the same processor.

Mapping Principle. If the home group of processor k consists largely of cells in the core or influence region of the cells mapped onto a processor 1, then the distance, $d_{k,l}$, should be as small as possible.

More formally, we first observe that interconnection networks can be readily partitioned into equal-sized subnetworks such that the average distance between two processors in a subnetwork is less than the average distance for the entire network. For example, an n-dimensional boolean hypercube can be subdivided into 2^q $(n - q)$-dimensional hypercubes. Moreover, the process is recursive in that the subnetworks can be further partitioned into smaller networks having smaller values for the average distance. In the partitioning and mapping scheme given below, care is taken to ensure that all partitions of the interconnection network have this property.

The interconnection network is first divided into subnetworks of approximate size $M/(G_i + G,)$ each. The cells that are mapped onto the processors of a subnetwork are in more frequently communicating cores or influence regions. If the core size is greater than home size, let $G_c = rM/N$, for some integer, r. A core is divided into r sets of equal or comparable sizes, which form the home groups for the processors in a subnetwork of size r within the larger subnetwork of size $M/(G_i + G,)$. Otherwise, we allocate all cells of a core to a single processor. This allocation scheme is consistent with the two principles given above and yields a good approximation to Eq. (3) without requiring extensive computation.

3.4. Packaging and VLSI Considerations

The memory capacity of a connectionist machine is measured by the number of cells and interconnects that can be stored. The speed of the machine is measured in the number of *interconnects* processed *per second* (IPS). Previously, some authors used the term *connection updates per second* (CUPS), which is equivalent to IPS if the network is trainable. In this section, we determine the amount of hardware required to build a connectionist system with a given capacity and speed.

Let H = M/N be the average size of a home group. To support networks with up to 2^{8n} cells and an average of C connections per cell, the home memory of each processor will require at least $n\,HC$ bytes to store the neighbor lists. If outputs and weights are 1 byte each, then the weight table requires at least another HC bytes. For networks up to 2^{24} cells and with C in the range $O(10^2 - 10^3)$, the size of the output table is much less than 4 HC bytes because many cells in the home group have common neighbors.

The network interface of processor k stores the list of virtual neighbors for each cell of the home group. The virtual neighbors of the processor k have buffers or bins allocated to them. The updated output of a cell is stored in the bins of its virtual neighbors. The capacity of the bin allocated to processor l is min (packet size, $v_{k,l}$). When this bin gets full, its contents are bundled into a packet and sent to l.

A total of about 6 HC bytes is required for the three memories and the neighbor lists and bins in the network interface. Less than 10% of this is due

to the remote section of the output table. If a single system-wide memory had been used, then the total size of the home memory and weight table would be about N times that in the local memory of a processor, but the replication of data in the remote output tables would have been avoided. We see that the increase in total memory due to the loosely coupled implementation with distributed memory is not very significant. The sizes of the output table and interface buffers scale almost linearly with the home size, H, provided H is not much less than G,. So the total memory required does not increase significantly with the system size until $H \ll G_c$.

For $N = 2^{12}$ and C = 1000, about 2 Mbytes of memory is required per processor if networks with up to 1 million cells are to be supported. This is reduced to around 128 Kbytes if the number of processor is increased to 64K. This amount of memory will dominate the real-estate requirements for a processor, since the pipelined arithmetic unit and other logic require much less area on silicon. So the size of a processor is inversely proportional to the network size, as the total memory required is almost constant for the range of network sizes under consideration.

Let us look at packaging considerations for building a system with 64K processors. In view of current technology, the following limits are assumed:

Max. processors/chip = 32 (area limited)

Max. I/O pins/chip = 256 (pin limited)

Max. chips/ board = 256 (area limited)

Max. I/O ports/ board = 2400 (connector limited).

Suppose each bidirectional link needs two I/O pins, namely serial in and serial out. Under the above limitations, a 4 X 8 mesh or a (5, 1)-net (either bus-based or cubelet-based) can be implemented on a chip. For the hypercube, however, only 8 processors can be implemented on a chip since a 4-cube will require 16 X 2 X 12 = 384 I/O pins, which exceeds the limit. At the board level, the area limitation is again more critical than pin/connector limitations for the torus. So a board can accommodate the maximum limit of 8K processors for a torus, so that all the processors require 8 boards. In the case of the hypernet, four (5, 2)-nets can be implemented on a single board, and 32 boards are required to accommodate all 64K processors. In contrast, only a 7-cube can be accommodated on a board, by barely meeting the I/O connector limit. The total number of boards required is 16 times, and the number of interboard connections is almost 18 times, that for the hypernet.

These packaging statistics are summarized in Table IIIa, where only the links required to make the connections specified by the topology are consid-

TABLE III
PACKAGING REQUIREMENTS FOR A CONNECTIONIST MACHINE WITH 64K PROCESSORS

Topology	Number of processors			No. of I/O pins or off-board connectors used		
	Mesh	Hypernet	Hypercube	Mesh	Hypernet	Hypercube
			(a) Using bit-serial channels			
Per chip	32 (8 X 4 mesh)	32 (one (5,1)-net)	8 (one S-cube)	48 pins	64 pins	208 pins
Per board	8192 (128 x 64 mesh)	2048 (four (5,2)-nets)	128 (one 7-cube)	768 connectors	2164 connectors	2304 connectors
Overall system	8 boards	32 boards	512 boards	6144	69,248	9×2^{17}
			(b) Using 4-bit wide channels			
Per chip	32 (8 X 4 mesh)	32 (one (5,1)-net)	2	192 pins	256 pins	240 pins
Per board	4096 (64 X 64 mesh)	512 (one (5,2)-net	16 (one 4-cube)	2048 connectors	2048 connectors	1.536 connectors
Overall system	16 boards	128 boards	4K boards	2'5	2^{18}	3×2^{20}

ered in determining the I / 0 pins or off-board connectors used. The low link complexity of the torus is reflected in its hardware demands. However, the hardware savings due to the interconnection network is not commensurate with the much greater bandwidth requirements as compared with the hypercube or hypernet. For the hypercube, pinout limitations become crucial at both chip and board levels, while for the hypernet, the limit on I/O wires becomes significant only at the board level. This results in the hypercube requiring much more hardware and further underscores the advantages of small-degree networks [45].

It should be noted that, if a smaller number of processors were used with more memory per processor, then the maximum number of processors per chip would be smaller. In that case, the pin limitations of the hypercube would not be so evident. For example, if we were allowed only 4 processors per chip, then a 12-dimensional hypercube would require 16 boards as compared to 4 boards for the torus or the hypernet. On the other hand, if a wider bus were used, the pin-out limitations would become even more critical, making high-degree topologies quite untenable for large systems. This is exemplified in Table IIIb, which is based on allocating 8 pins per input / output channel pair.

4. Efficiency of Neural Net Simulation

Highly parallel simulation of large neural networks is restricted by the interprocessor communication required to transmit the activation level of each cell to its neighbors. In this section, we estimate the minimum bandwidth required of a multicomputer interconnection network so that interprocessor communication does not become a bottleneck during the execution of a neural network specified by the general model of Section 2.2. First, theoretical estimates are given. Experimental results are presented and compared with the theoretically obtained values. The effect of the structure of a neural network on communication demands is shown. Hypercubes, hypemets, and toruses are evaluated with respect to their ability to handle the desired volume of interprocessor message traffic. Finally, the effectiveness of broadcasting messages is examined.

4.1. Communication Bandwidth Requirements

An efficient simulation is based on a balanced utilization of all resources: processors, memory, interconnection network, and system I/O. For our purposes, the crucial balance is that between processing and communication capabilities. So we need to estimate the bandwidth required in terms of the average number of packets that a processor handles per iteration cycle. This

metric indicates the bandwidth required of the physical communication channels to sustain a given computational rate. Alternatively, given the specifications of an interconnection network, it yields the number of iteration cycles that can be reasonably executed per second for a particular problem. A packet-switched network using a store-and-forward routine for packets in transit and infinite buffer storage is considered.

The packets handled by a processor are those for which the processor is either the origin or the final destination, as well as the packets which reach their final destination via that processor. Let Φ_i be the average number of message packets that are sent by processor k per iteration cycle to simulate the connections between its home group and those cells that are in the influence region of this group. We define Φ_c and Φ_r in a similar fashion to cater to the connections in the core region and the remote region. We first estimate Φ_i. Let $\rho_c = C_c/G_c$ be the probability that two cells within a core are connected, and H = M/N be the size of a home group. Following the mapping policy given in Section 3.3, we have two cases:

Case 1. $G_c \geq H$.

Here, the home group of processor k consists solely of cells belonging to the same core. Consider a cell a in this home group which is connected to a cell b in its influence region. Let cell b be allocated to processor $l, l \neq k$. If the connections to the influence region were randomly distributed, the average number of connections between cell a and the cells mapped onto processor 1 is max(1, HC_i/G_i). However, due to the clustering effect, the probability of a connection between cell a and a cell in I increases to $\rho_c\rho_i$, if ρ_i is high enough. Thus, the average number of connections between cell a and a virtual neighbor is about max($\rho_c\rho_i H, 1, HC_i/G_i$). These connections can be simulated by a ***single*** message to the neighbor for every output update. Thus, the average number of messages sent per cell, which is the same as the average number of virtual neighbors of a cell, $= C_i/\rho H$, where

$$\rho = \frac{H}{\max(\rho_c\rho_i H, 1, HC_i/G_i)}. \tag{4}$$

To find the number of packets sent by processor k, we first determine the expected value of $v_{k,l}$. If ρH = max(1, $C_i H/G_i$), then we can assume the connections to be randomly distributed, so that the average value of $v_{k,l}$ is $H(1 - e^{-HC_i/G_i})$, as deduced later (see Eq. (9)). Otherwise, let E be the set of cells on processor *1* which are neighbors of cell a, and Fbe the set of home neighbors of cell a. If e $\in E$, and $f \in F$ are chosen at random, then the probability that there is no connection between e andf is $1 - \rho_i$. Cell a has approximately $\rho_c H$ neighbors on its home processor, and ρH neighbors on processor *1*. Thus, the probability that f has a neighbor in set F is

$1 - (1 - \rho_i)^{\rho H}$, which is approximately $1 - e^{-\rho\rho_i H}$ if $\rho_i \ll 1$. Furthermore, if $\rho_c H[1 - e^{-\rho\rho_i H}] \gg 1$, then we can assume that every cell in processor k's home group has a home neighbor that is also a neighbor of processor l. Then

$$v_{k,l} = H[1 - e^{-\rho\rho_i H}]. \tag{5}$$

Therefore, the average number of packets sent per processor is

$$\Phi_i = \left\lceil \frac{H[1 - e^{-\rho\rho_i H}]}{\beta} \right\rceil_1 \times \frac{C_i}{\rho H[1 - e^{-\rho\rho_i H}]}, \tag{6}$$

where ρ is given by Eq. (4), and β is the maximum number of messages per packet, as stated earlier.

Case 2. $rG_c = H, r > 1$.

Again, let 1 be the home processor for a cell b that is connected to cell a and is in its region of influence. Besides the core of b, the home group of processor 1 includes $r - 1$ other cores. All these groups are in the influence region of one another, and by transitivity, in the influence region of cell a. However, the existence of the connection to cell b does not affect the probability of cell a's having a connection with a cell in one of the other cores on processor 1. Therefore, the average number of connections from a to the home group of processor l is

$$\frac{H}{r}\left(\rho_c\rho_i + \frac{(r-1)C_i}{G_i}\right).$$

The first term is the number of connections to the core of b, while the second is the expected connects to the $r - 1$ other cores that share the same home group. The number of messages sent by a cell is $C_i/H\rho'$, where

$$\rho' = \max\left[1, \frac{C_i H}{G_i}, \frac{1}{r}\left(\rho_c\rho_i + \frac{(r-1)C_i}{G_i}\right)\right].$$

With an analysis similar to that for Case 1, we can approximate the average number of messages reaching a virtual neighbor 1 of processor k by

$$v_{k,l} = H[1 - e^{-\rho'\rho_i H}] \quad \text{if} \quad \rho' H[1 - e^{-\rho'\rho_i H}] > 1 \quad \text{or} \quad \frac{\rho_c C_i H}{G_i} > 1,$$

$$v_{k,l} = \frac{H[1 - e^{-\rho'\rho_i H}]}{r}\left[1 + \frac{(r-1)C_i\rho_c H}{G_i}\right]_1 \quad \text{otherwise.} \tag{7}$$

The average number of packets sent per processor is

$$\Phi_i = \left[\frac{v_{k,l}}{\beta}\right] \times \frac{C_i}{\rho' v_{k,l}}, \tag{8}$$

where $v_{k,l}$ is given by (7) . Equations (6) and (8) give the average number of packets sent per processor to simulate those connections of its home group that go to their regions of influence. From the mapping scheme, we see that the processors that receive these packets are randomly distributed in a subnetwork of size NG_i/M that includes processor k.

A processor also sends Φ_r message packets to processors outside this subnetwork to cater to the connections of its home group to the "remote" region. The value of Φ_r can be estimated in the same way as that for Φ_i, by substituting ρ_r for ρ_i, G_r for G_i, and C_r for C_i in Eqs. (6) and (8).

An estimate for Φ_c is obtained more simply. For Case 2, all core connections are internal to a processor. So no packets need to be sent to other processors, that is, $\Phi_c = 0$. In Case 1, a core is spread over $\lceil G_c/H \rceil$ processors. If $\rho_c H \gg 1$ then, with high probability, all the other $\lceil G_c/H \rceil - 1$ processors are virtual neighbors for a cell in any one of these processors. Therefore,

$$\Phi_c = \frac{H(\lceil G_c/H \rceil - 1)}{\beta}.$$

For reference, let us now estimate the number of packets sent to processors in the influence region if ρ_i was not given and instead we assumed that the connections to this region were distributed randomly. We call this the *unconditional* case. It gives an upper bound for the number of packets and also indicates the effect of ρ_i in reducing interprocessor communication.

The influence region of a cell is spread among G_i/H home groups. Since a cell has C_i connections to this region, the probability that it has no connections to a specific home group is $(1 - H/G_i)^{C_i}$, which can be approximated by e^{-HC_i/G_i} if $G_i/H \gg 1$. Thus the average number of processors that are virtual neighbors of a cell because of its connections to the influence region is

$$\frac{G_i}{H}(1 - e^{-HC_i/G_i}).$$

Since the connections are random, the number of cells in processor k that have processor l as a virtual neighbor is simply given by

$$v_{k,l} = H(1 - e^{-HC_i/G_i}). \tag{9}$$

Therefore, the average number of packets sent by processor k is

$$\Phi_i = \left\lceil \frac{v_{k,l}}{\beta} \right\rceil \frac{G_i}{H} \qquad \text{for} \quad v_{k,l} \geq 1,$$
$$\Phi_i = G_i(1 - e^{-HC_i/G_i}) \qquad \text{elsewhere.} \tag{10}$$

The value of Φ_r for the unconditional case is calculated in a similar way and can be calculated by simply substituting G_r for G_i and C_r for C_i in Eq. (10).

The average number of packets handled by a processor depends on the multicomputer topology used. Let d_c be the average distance (in terms of the number of links traversed from source to destination) to a virtual neighbor in the core region. Similarly, we define d_i and d_r for virtual neighbors in the influence region and the remote region, respectively. Then the average bandwidth required per processor per iteration cycle is

$$\Phi_c(d_c + 1) + \Phi_i(d_i + 1) + \Phi_r(d_r + 1). \tag{11}$$

To estimate the average distance to a processor in a particular region, we first observe that the processors in a specific region are essentially confined to a subnetwork of size M/G_{region}. By assuming the destination to be randomly placed within this subnetwork, we get a conservative value of the average distance. Consider a subnetwork of size 2^q. If this network is a boolean hypercube, the average distance is $q/2$. In the case of a torus the average distance is given by

$$0.5 \times (1 - 2^{q/2}) \qquad \text{if } q \text{ is even ;}$$
$$0.75 \times (1 - 2^{(q-1)/2}) \qquad \text{if } q \text{ is odd.}$$

For a $(d,$ h)-hypernet constructed from cubelets, this distance is bounded above by $2^{h-2}(d + 2) - 1$. If the hypernet is constructed from buslets, the distance is reduced to $2^h - 1$ provided there is no bus contention. Thus, given the number of processors in each of the three regions and the multicomputer used, we can calculate d_c, d_i, and d_r. Since we already know how to estimate Φ_c, Φ_i, and Φ_r, we can now evaluate Eq. (11).

4.2. *Analysis of Simulation Results*

Table IV gives the specifications for the network graphs of six hypothetical neural systems that were used for simulation. The descriptions are in accordance with the structural model described in Section 2, except that ρ_i and ρ_r are not specified. Nets A to D have about 16 million cells each. The average

TABLE IV
STRUCTURAL DESCRIPTION OF VARIOUS NETWORKS

Name	M	G_c	G_i	C_c	C_i	C_r
Net A	2^{24}	512	65,536	128	128	32
Net B	2^{24}	256	16,384	64	128	128
Net C	2^{24}	512	8,192	256	256	16
Net D	2^{24}	2048	65,536	1024	1024	512
Net E	2^{20}	128	8,192	128	128	32

number of connections per cell ranges from 288 (net A) to 2500 (net D). Net A serves as a reference structure, compared to which the connections of net B are more spread out while the connections of net C are more concentrated in regions of locality. Net D is characterized by almost an order of magnitude greater number of connections as compared to the other nets. Finally, net E has a structure similar to that of net A but has only 1 million cells.

To obtain values of Φ_c, Φ_i, and Φ_r through simulation, an instance of each net is first generated by getting the values of the model parameters from a normal distribution with standard deviation, σ, equal to a tenth of the mean values given in Table IV. Each net is thus represented as a "macronetwork" consisting of interconnected cores. The values of ρ_i and ρ_r are then used to obtain the distribution of connections between these cores. This results in a network described by a list of individual core sizes and the number of connections between each pair of cores. These cores are mapped onto hypercube, hypernet, and torus architectures using the principles given in Section 3.3. The number of packets that need to be handled per processor per iteration is then determined from the average number of connection paths going through a processor.

The *packet size,* given by the (maximum) number of messages sent in a packet, is 16, unless mentioned otherwise. As mentioned in Section 3.2, a packet contains a cell identity number and an output value for each message that it conveys. In addition, it carries header information containing the addresses of the source and destination processors, and control bits. Thus, the size of a full packet is about 50 bytes.

4.2.1. *Effect of Clustering*

First, we mapped net A on a 14-dimensional hypercube and calculated the bandwidth required for various values of ρ_i and ρ_r. In the unconditional case where the connections within each region are distributed randomly, the average number of packets handled per processor is 265,2 16, which corresponds to over 16 Mbytes/iteration cycle. Figure 6 shows the number of

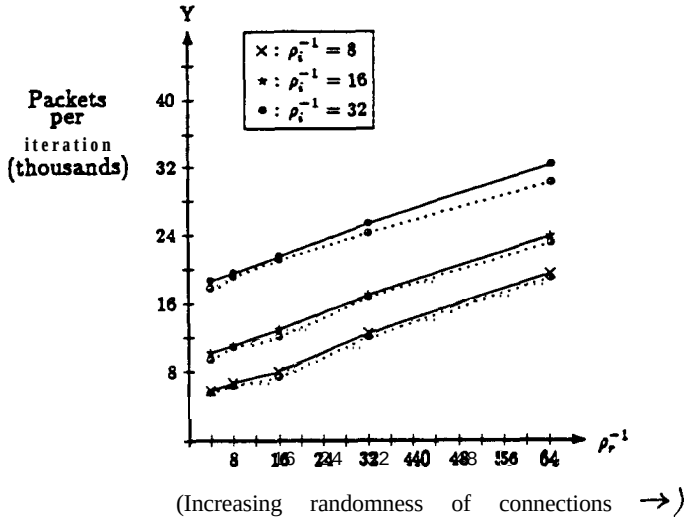

FIG. 6. Effect of clustering on bandwidth requirements.

packets handled for different values of the conditional probabilities. The dashed lines correspond to the theoretical estimates, while the solid lines are the experimentally observed values. We see that for net A, ρ_i is somewhat more critical than ρ_r in determining the bandwidth. If both ρ_i and ρ_r are greater than 1 / 16, then the bandwidth required is reduced to well within 1 Mbyte/iteration cycle, which is less than 5% of the unconditional case. This indicates that prominent clustered features in the structure of a neural network are critical in bringing bandwidth requirements to a manageable level. We also observe that the theoretical estimates are more conservative. The experimental values obtained are from 2 to 9% greater than the theoretical predictions.

4.2.2. *Effect of System Size*

Figure 7a shows how the bandwidth varies with the number of processors in the system. The curve is based on the mapping of net B with $\rho_i = 1 / 8$ and $\rho_r = 1 / 16$, onto hypercube sizes ranging from 1024 to 256K processors. The increase in bandwidth required when the system size is smaller than 64K is attributable to two factors:

- The size of a home group increases. A processor has to send more messages to its virtual neighbors. This factor is not so dominant because a cell needs to send at most one message to a processor irrespective of the number of cells in that processor's home group to which this cell is connected.

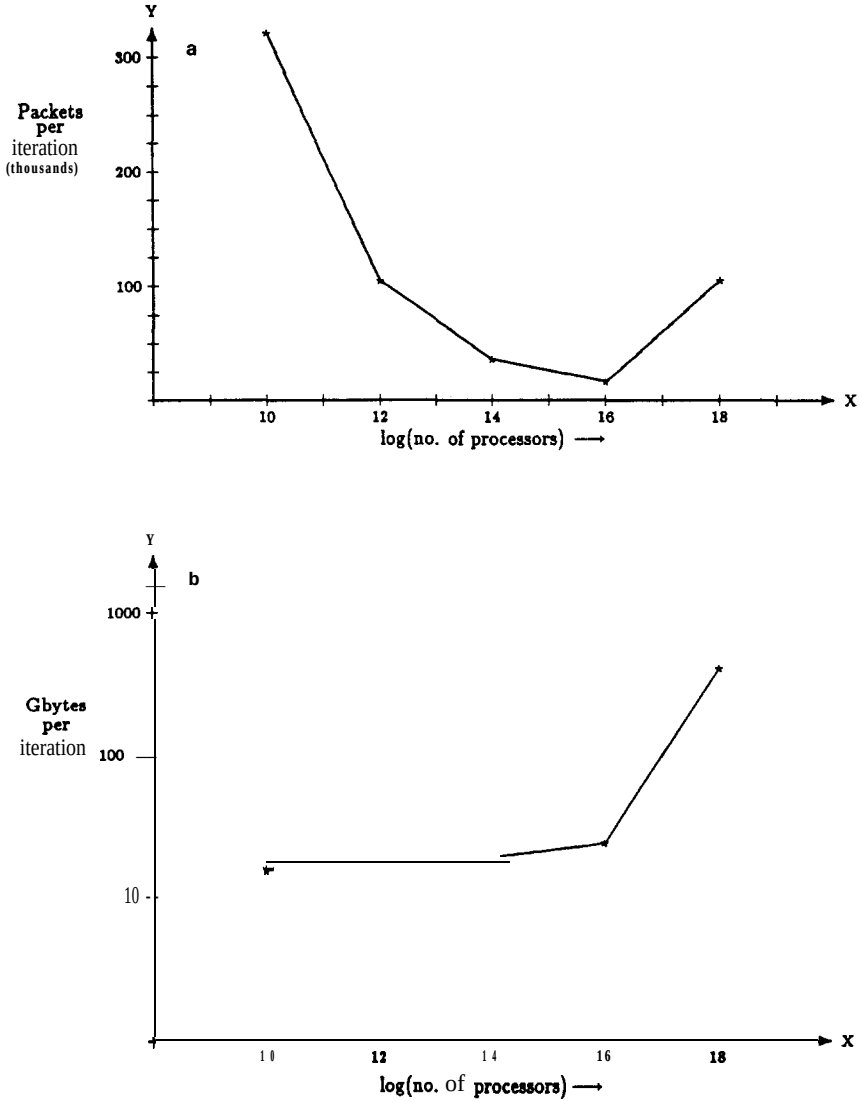

FIG. 7. Bandwidth demands for different system sizes. (a) Per processor. (b) Overall system.

• When more than one core is mapped onto a processor, the impact of clustering is diluted because some external core may have many connections to one core but very few to the other cores in the same home group. This is largely responsible for the 20-fold increase in bandwidth for 1024 processors as compared to the "optimal" system size of 64K.

Interestingly, the bandwidth requirement shoots up dramatically when the system size is increased beyond 64K, despite the presence of smaller home groups. This is mainly due to three factors: First, the cores are now spread over several processors, and so messages to simulate core connections also need to be sent. Second, a cell has very few connections going to a neighboring processor. Since the total number of connections per cell is the same, the number of messages sent by a cell increases. Finally, $v_{k,l}$ is very small due to the small home sizes. Packets are almost empty. This is particularly significant for packets sent to the remote region where the average number of messages conveyed per packet was found to range from 1.01 for net C to 7.52 for net D when 256K processors were used. These factors again weaken the clustering effect. The message patterns approach those of the unconditional case as the system size approaches M.

The total number of packets handled in the system per iteration cycle of course is expected to increase with the system size. This is verified in Fig. 7b. This means that the total number of communication demands increases faster than the speedup achieved by adding more processors to the system, provided all processors have the same computational power. For example, suppose a processor can process 1 million connections per second. Neglecting overheads, one iteration of net B will take about 1.25 sec on a system with 4K processors. This demands a total communication bandwidth of 3.2 Gbytes/sec. If we use 64K processors, then one iteration takes only 0.078 sec, provided the bandwidth is 18.5 Gbytes/sec. The rapid increase in required total bandwidth with the system size is brought out in Fig. 8, where the bandwidth required to achieve a linear speedup is shown as a function of the system size, when net B (with $\rho_i = 1/8$, $\rho_r = 1/16$) is mapped. It is assumed that each processor has a speed of 1 MCUP.

4.2.3. *Effect of Network Structure*

Figure 9 shows the bandwidths required by the other five nets for the same values of ρ_i and ρ_r, when mapped onto hypercubes of various sizes. All these nets need less bandwidth than net B. This is particularly remarkable for net D, which has many more connections than net B. It was experimentally observed that for net D, more packets were sent in the influence region. However, the number of packets sent to the remote region was significantly less as compared to that for net B. This counterintuitive observation is due to two effects. First, the number of messages sent per cell was actually less by 46 to 77% for net D because the denser cores of D caused the clusterings to be more prominent. Second, $v_{k,l}$ was larger for net D because of the denser cores and increased number of connections. Consequently each processor had fewer virtual neighbors on the average when net D was mapped. In general, the sparseness and spread of connections are seen to be more demanding on the system bandwidth than the actual number of connections.

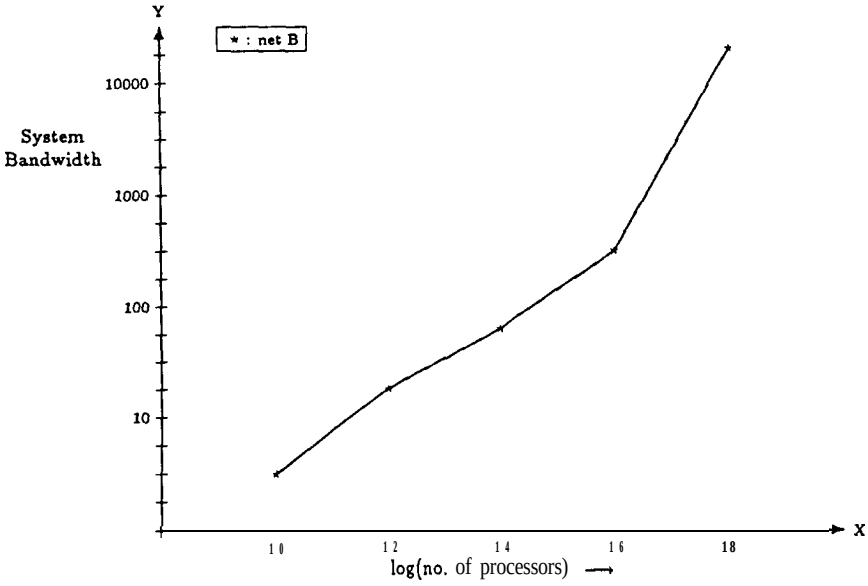

FIG. 8. Interprocessor communication demands for linear speedup.

Some other observations made from Fig. 9 are

- For all curves, the minima correspond to system sizes for which the home group and the core sizes become comparable.
- Net D has a structure similar to that of net A but almost 10 times as many connections. However, the bandwidth requirements do not increase proportionally. This reinforces our earlier observation that the distribution of connections can be more critical than the actual number of connections for determining the system bandwidth.
- Net C requires the least amount of interprocessor communication because of the greatly localized nature of its connections.
- Net E has a structure similar to that of net A but contains only 1/16th the number of cells. This does not cause a significant reduction in the bandwidth demands, though there is a prominent shift in the optimum system size.

4.2.4. *Influence of Multicomputer Topology*

To examine the effect of the multicomputer topology on bandwidth demands, net A (with $\rho_i = 1/8$, $\rho_r = 1/16$) was mapped onto different architectures. The **hypernet** used for comparison is built from **5-cubelets** [36]. The *busnet* is a particular type of **hypernet** constructed using **5-buslets**. In the

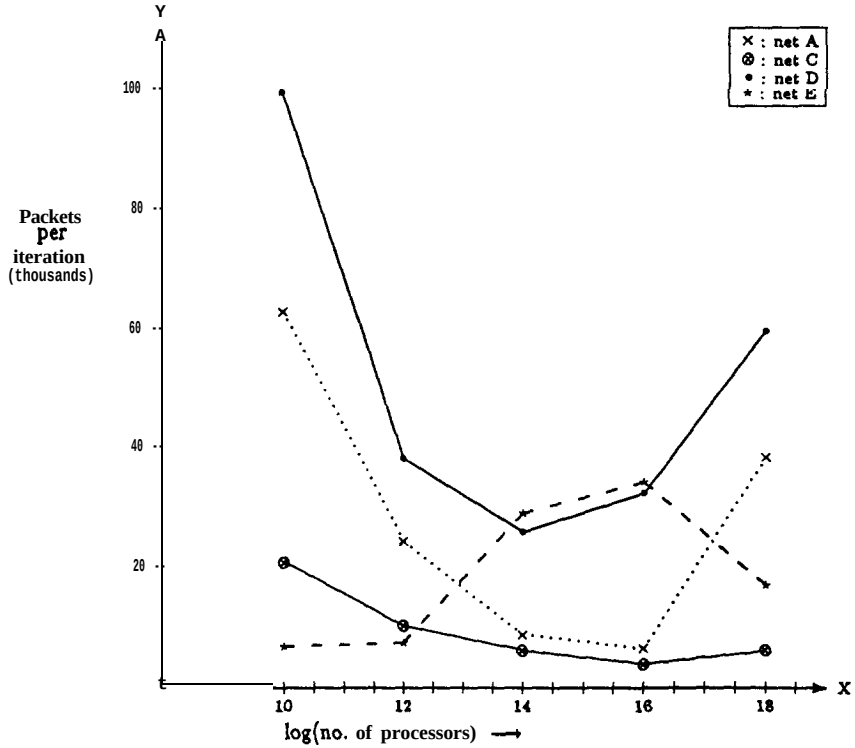

FIG. 9. Bandwidth required for different nets.

simulation of a **buslet,** an ethernet type of protocol was assumed with a 25% chance that a packet would be accepted. The results are shown in Fig. 10a.

Not surprisingly, hypercubes, which have the highest degree of the four topologies, have to handle the least number of packets. However, **hypernets** are seen to have a comparable performance while using fewer communication links per processor. The torus suffers from a large diameter $(O(\sqrt{N}))$, and so performs poorly, particularly when the network size is large.

For a meaningful comparison between networks with different numbers of output ports per node, we normalize the bandwidth demanded by multiplying it by the number of ports per node [6, 27]. This is based on fixing the total bandwidth available per node, so that the bandwidth available per port is inversely proportional to the number of ports for that node. When this is done, **hypernets** are seen to be more effective than hypercubes, as shown in Fig. 10b. The advantages of networks with low link complexity are further accentuated when the finite size of message buffers and VLSI layout considerations are also taken into account [11,451.

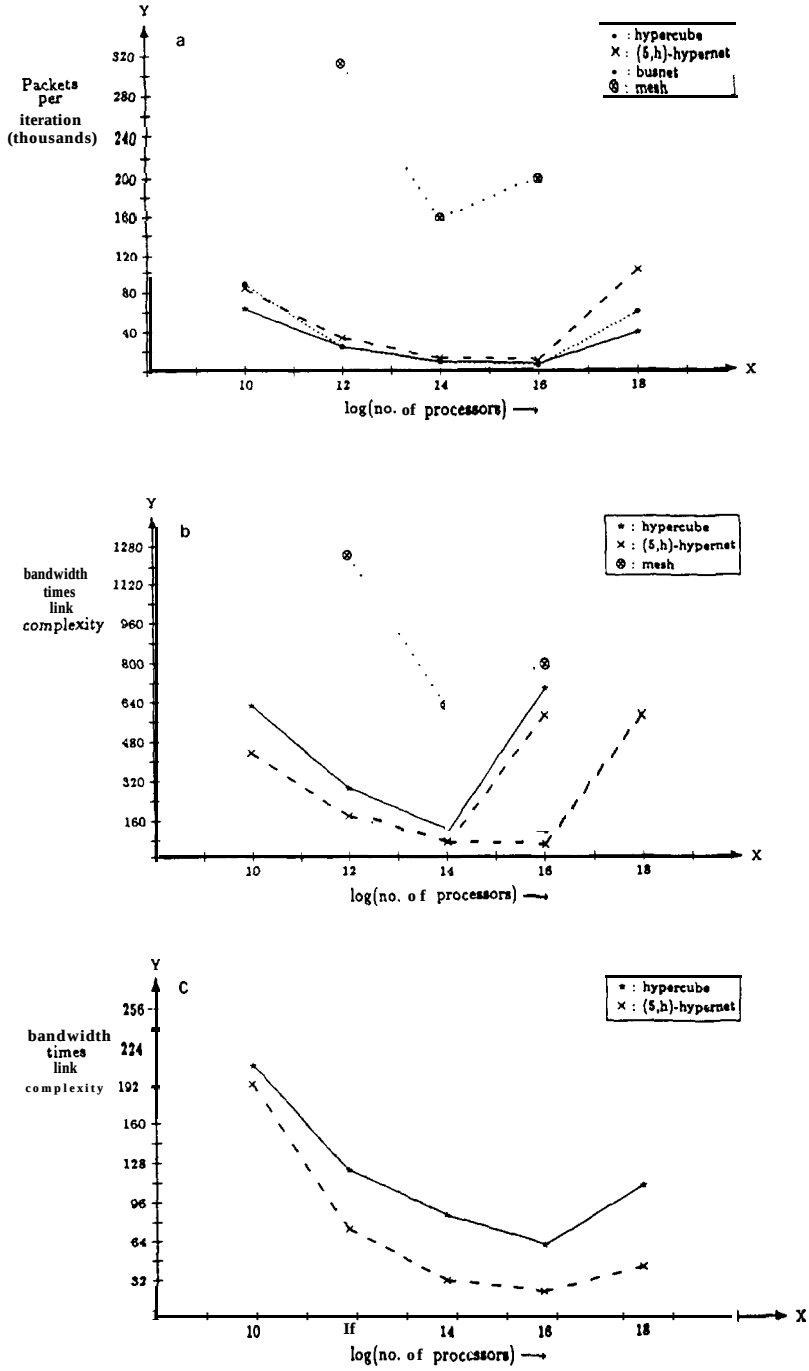

FIG. 10. Bandwidth requirements for different multicomputer topologies. (a) Bandwidth demands. (b) Normalized bandwidth demands. (c) Bandwidth requirements for net C (normalized).

Since hypernets provide more support for local communication while maintaining a constant degree, they are more suited for implementing networks in which most of the connections from a cell are confined to a small subnetwork. This can be seen readily from Fig. 10c, which shows the normalized bandwidth (for $\rho_i = 1/8$ and $\rho_r = 1/16$) when net C is simulated. We conclude that hypernets are more cost effective in supporting interprocessor communications when the neural models being implemented have a highly localized or clustered structure.

4.2.5. Broadcast Options

In the previous section, the outputs of a cell were conveyed to its virtual neighbors through personalized packets. An alternative is to broadcast the output to all processors (total broadcast) [2 1], or to the processors in a predefined neighborhood called the ***broadcast domain*** (partial broadcast) [22].

On receiving a broadcast packet, a processor examines the sender's address and records only those messages that are needed by its home group. Other messages are simply ignored. Optimum distributed broadcast schemes have been proposed for the hypercube, hypernet, and torus [3 1, 36]. For the first two architectures, a broadcast message is received by all processors in 0(log N) time. For the torus, $0(\sqrt{N})$ time is required. In all three schemes, each node is visited only once.

Broadcasting is more efficient than sending personalized messages if, on the average, a sizable number of processors in the broadcast domain are actually virtual neighbors of the sender cell. Since the percentage of virtual neighbors is very low in the remote region, it is not advisable to broadcast over the entire network. Total broadcasting requires a processor to handle about M/β packets per iteration. This is because a processor needs to send H/β packets per iteration, and each packet goes to $M/H \sim 1$ processors. For a network with 2^{24} nodes and a packet size of 16, this translates to over 1 million packets per processor per iteration cycle. This far exceeds the requirements for personalized communication for all the networks considered. On the other hand, broadcasting to processors implementing the same core region (for Case 2) always shows an improvement.

Broadcasting to all processors implementing an influence region is preferable if ρ_i is less than some threshold. It is shown in [24] that broadcasting to this region is surely less demanding when

$$\rho_i < \frac{C_j d_i}{\rho_c G_i}. \tag{12}$$

4.3. Simulation Efficiency of Existing Parallel Computers

The suitability of existing parallel computers for simulating neural networks can be evaluated on the basis of the architectural requirements deter-

mined by the simulation efficiency analysis. Our analysis suffices to detect bottlenecks and imbalances between processing, memory, and communication requirements. A more thorough assessment would require examining machine characteristics and implementation strategies in great detail, together with a more intimate analysis of the neural network structure.

Consider the simulation of a neural net with 1 million cells and an average of 1000 connections per cell. Such a network will come close to attaining the computational capabilities of a cockroach if executed at over 20 iterations/sec [12]. Table V presents typical requirements *per processor* for simulating this network, given $\rho_i = 1/8$ and $\rho_r = 1/16$.

On comparing the specifications of existing multiprocessors, such as the Intel iPSC, NCUBE/ten, Ametek/14, and BBN Butterfly, with this table, we find that medium-grained systems suffer from inadequate memory and communication bandwidth to fully exploit their processing capabilities. For example, the NCUBE/ten offers a configuration with 1K processors. Using noninteger operations, this system could simulate the network at almost 2 iterations/sec only if there was enough memory to store the network. A BBN Butterfly Plus [2] with 256 processors falls short in memory storage as well as processor speed and communication bandwidth for the magnitude of the network being considered. The diversity of these complex-instruction-set-computer (CISC) based systems is not utilized fully since processing is largely confined to a fast execution of matrix-vector products which form the inner loop for neural computations. Transputer-based systems, such as the Meiko Computing Surface with 130 transputers [18], also suffer from the fact that their memory capacity and communication bandwidth are not commensurate with processing speed for neural network simulations.

One way to solve the memory capacity problem is to have a larger number of processors so that less memory is required per node. On examining fine-grained systems such as the CM-2 and the DAP, however, we find that the lack of local memory is even more critical for these machines. For example, in CM-2 each processor has only 8 Kbytes of local memory [3]. So we are

TABLE V
ARCHITECTURAL REQUIREMENTS FOR SIMULATING A NEURAL NETWORK WITH 1 MILLION CELLS

System size (No. of processors)	Memory (Mbytes)	Communication bandwidth (Mbytes/iteration)		Processor speed (M interconnects per iteration)
		(Hypercube)	(Torus)	
1,024	8	**0.5**	4	1
4,096	2	0.5	8	0.25
16,384	**0.5**	2	32	0.07

forced to use secondary storage, which leads to a new bottleneck in I/O transfers. For the 1 -million-cell network, the processors are capable of completing 1 iteration/sec, but the I/O bottleneck restricts speed to over 14 sec per iteration. On the other hand, if infinite I/O access speed was available, then the interprocessor communication abilities would have restricted speed to about 10 sec per iteration. Thus the I/O transfer rate is the more critical limitation.

We feel that transputer-based systems, massively parallel machines such as CM-2, and the new generation of message-passing multicomputers such as the Ametek 2010 [47] still offer the best alternatives for highly parallel neural simulation. The first has the advantages of high-speed concurrent communication links, hardware support for process scheduling, and a large addressable memory space. However, forwarding of messages through intermediate nodes can be achieved only through software routines. This slows down communication between nonadjacent processors and limits its viability to systolic implementations [4 1].

For massively parallel machines, a method of allocating one processor for each cell and two processors for each weight has been suggested recently by Blelloch and Rosenberg [9]. This spreads the weights of the connections of any cell over several processors. These weights can be efficiently processed using scan operations and pipelining. Using such a representation, the CM-2 is able to make up to 64M interconnects.

For the Ametek 2010, a configuration of 1024 mesh-connected nodes would have just enough memory to store the target network and execute it at 1 iteration/sec. At this speed, the interconnection network bandwidth is not a bottleneck since the "wormhole routing" mechanism incorporated in the 2010's automatic message-routing device enables each node to handle over 20 Mbytes/sec per channel with little overhead [47]. Furthermore, the high-resolution graphics display can provide a visual trace of the network execution.

An orthogonal approach is to have a much smaller number of extremely powerful numerical processors communicating over a common bus. This reduces the communication requirements while providing enough processing power and storage capacity for moderately large neural nets. A pipelined architecture with a limited instruction set or the use of fast multipliers is appropriate for the processor nodes [40]. A notable effort in this direction is the MX-I / 16 being developed at MIT Lincoln Laboratories [26]. This multiprocessor is intended for AI applications such as machine vision that involve both numeric and symbolic computations. It comprises 16 digital signal processors, each capable of 1 OM multiply-accumulate operations/ sec, interconnected through a crossbar to one another and to a host Lisp machine. The system aims for rapid prototyping of parallel algorithms and their evaluation using large data bases.

In conclusion, a fully configured Ametek 20 10 or the prototype MX- 1 / 16 will provide enough storage and processing power for meeting today's neural

network simulation needs but still fall far short of the computational capabilities of modest biological systems. This provides motivation for the long-term development of specialized computers for highly concurrent simulation of large neural networks.

5. Conclusions

This paper presents a distributed processor/memory organization for highly concurrent simulation of neural networks. On the basis of the structural model and mapping policy, we estimated the interprocessor communication bandwidth required to balance processing and communication demands. The interconnect patterns among the cells have a dramatic impact on the bandwidth requirements. A moderate amount of clustering can reduce the bandwidth demands by over 90% when compared to a totally random interconnect pattern. Since the communication bandwidth is expected to be the principle bottleneck in a highly parallel implementation, this provides an incentive for developing connectionist nets with local and clustered interconnects.

We reason that architectures with direct links or multiple buses for interprocessor communication are preferable to multistage networks. Topologies such as hypernets and cube-connected cycles are considered suitable for designing a concurrent connectionist machine as they have low link complexity and are able to provide support for both localized and global communication patterns.

Instead of sending personalized messages between processors, one can use broadcasting techniques. It is seen that broadcasting messages throughout a large system is very expensive. However, a partial broadcast to the influence regions leads to less communication when Eq. (12) is satisfied. This suggests that the interconnection network can be profitably augmented by adding local buses or fan-out trees. Alternatively, one can use architectures such as bus-based hypernets that employ both local buses and direct communication channels.

Even net C, which has the lowest requirements of the nets considered with $\rho_i = 1/8$, $\rho_r = 1/16$, needs a total bandwidth of over 6 GBytes/sec when mapped onto a 16-dimensional hypercube, in order to keep the iteration cycle time within 1 sec. Implementation of large networks also motivates the search for new technologies, such as optics, that can provide very high bandwidths. Issues involved in using optical interconnects for constructing a neural net simulator are explored in Ghosh and Hwang [25].

An alternative is to reduce bandwidth demands by modifying the computational model. For example, a cell may broadcast its updated output only if it differs significantly from its old output. Moreover, it may be allowed to update its state even when it has not received the updated outputs of many

of its neighbors. The effect of these pragmatic policies on execution speed, and indeed on the results themselves, is an area not fully explored yet [22]. Other important areas of research include specification of network models to solve a given problem, effective knowledge representation and techniques for network compilation, run-time modifications to cater to new knowledge, extraction of knowledge and results from the networks, and the design of user interfaces [17, 19]. Significant progress in all these areas is needed to realize the full potential of connectionist computation.

REFERENCES

1. *ANZA Plus VME.* HNC Inc., 5501 Oberlin Drive, San Diego, CA 92121-1718, 1988.
2. *Butterfly Products Overview.* BBN Advanced Computers Inc., Oct. 14, 1987.
3. *Connection Machine Model CM-2 Technical Summary.* Thinking Machines Corp., 1987.
4. *Z-Series: Workstations for Artificial Neural Systems,* Science Applications International Corporation, 10260 Campus Point Drive, San Diego, CA 92 12 1, 1987.
5. *The TRW Mark III-l Artificial Neural System Processor.* TRW MEAD AI Center, San Diego, CA, 1987.
6. Agrawal, D. P., Janakiram, V. K., and Pathak, G. C. Evaluating the performance of multicomputer configurations. *IEEE Comput.* **19, 5** (May 1986), 23-37.
7. Amari, S. Competitive and cooperative aspects in dynamics of neural excitation and self-organization. In Amari, S., and Arbib, M. A. (Eds.). *Competition and Cooperation in Neural Nets.* Springer-Verlag, New York/Berlin, 1982.
8. Arbib, M. A., and Hanson, A. R. *Vision, Brain and Cooperative Computation.* MIT Press, Boston, 1987.
9. Blelloch, G., and Rosenberg, C. R. Network learning on the Connection Machine. *Proc. 10th International Joint Conference on Artificial Intelligence,* Milan, Italy, Aug. 1987, pp. 323-326.
10. Cohen, M. A., and Grossberg, S. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. Systems, Man Cybernet.* **SMC-13,** 5 (Sept./Oct. 1983), 815-826.
11. Dally, W. J., and Seitz, C. L. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput. C-36,5* (1987), 547-553.
12. DARPA. *Executive Summary of DARPA Neural Network Study.* MIT Lincoln Labs., Lexington, MA, July 8, 1988.
13. Edelman, G. M., and Mountcastle, B. *The Mindful Brain.* MIT Press, Boston, 1978.
14. Fahlman, S. E., and Hinton, G. E. Connectionist architectures for artificial intelligence. *IEEE Comput.* (Jan. 1987), 100-109.
15. Fahlman, S. E., and Hinton, G. E. Massively parallel architectures for AI: NETL, Thistle and Boltzmann Machines. *Proc. National Conference on Artificial Intelligence, 1983,* pp. 109-l 13.
16. Feldman, J. A., and Ballard, D. H. Connectionist models and their properties. *Cognitive Sci. 6* (1982), 205-254.
17. Feldman, J. A., Fanty, M. A., Goddard, N. H., and Lynne, K. J. Computing with structured connectionist networks. *Comm. ACM* **31, 2** (1988), *170-187.*
18. Forrest, B. M., Roweth, D., Stroud, N., Wallace, D. J., and Wilson, G. V. Implementing neural network models on parallel computers. *Comput. J.* **30, 5** (1987), 413-419.
19. Gallant, S. I. Connectionist expert systems. *Comm. ACM* **31, 2** (1988), *152-169.*

20. Gamble, E., and Poggio, T. Integration of intensity edges with stereo and motion. Tech. Rep., AI Lab Memo No. 970, MIT, Cambridge, MA, 1987.

2 I. Garth, S., and Pike, D. An integrated system for neural network simulations. *Comput. Architect. News* 16, 1 (1988), 37-44.

22. Geller, R. D., and Hammerstrom, D. W. A VLSI architecture for a neurocomputer using high-order predicates. *Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence,* 1987, pp. 153– 16 I.

23. Geman, S. Notes on a self-organizing machine. In Hinton, G. E., and Anderson, J. A. (Eds.). *Parallel Models of Associative Memory.* Erlbaum, Hillsdale, NJ, 198 1, pp. 237-263.

24. Ghosh, J. Communication-efficient architectures for massively parallel processing. Ph.D. thesis, University of Southern California, Los Angeles, Apr. 1988.

25. Ghosh, J., and Hwang, K. Optically connected multiprocessors for simulating artificial neural networks. *SPIE Proc. 882* (Jan. 1988).

26. Goblick, T., Harmon, P., and Leivent, J. A multiprocessor for mixed numeric/symbolic computation. Talk given by MIT Machine Intelligence Group, Oct. 4, 1988.

27. Goodman, J. R., and Sequin, C. H. Hypertree: A multiprocessor interconnection topology. *IEEE Trans. Comput. C-30,* 12 (Dec. 198 I), 923-933.

28. Hammerstrom, D., Maier, D., and Thakkar, S. The cognitive architecture project. *Comput. Architect. News 14* (1986), 9-21.

29. Hecht-Nielsen, R. Performance limits of optical, electro-optical, and electronics neurocomputers. *Proc. SPIE* 634 (1986), *277-306.*

30. Hillis, W. D. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

3 I. Ho, C. T., and Johnsson, S. L. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. *Proc. International* Conference *on Parallel Processing* (Aug. 1986), pp. 640–648.

32. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. U.S.A.* 79 (Apr. 1982), 2554-2558.

33. Hopfield, J. J., and Tank, D. W. Computing with neural circuits: A model. *Science* (Aug. 8, 1986), 625-633.

34. Hutchinson, J., Koch, C., Luo, J., and Mead, C. Computing motion using analog and binary resistive networks. *IEEE Comput.* 21, 3 (1988), 52-63.

35. Hwang, K., Chowkwanyun, R., and Ghosh, J. Parallel architectures for implementing AI systems. In Hwang, K., and DeGroot, D. (Eds.). *Parallel Processing for Supercomputers and Artificial Intelligence.* McGraw-Hill, New York, 1989.

36. Hwang, K., and Ghosh, J. Hypernet: A communication-efficient architecture for constructing massively parallel computers. *IEEE Trans. Comput. C-36 (Dec.* 1987), 1450–1466.

37. Kernighan, B., and Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell Systems Tech. J.* 49, 2 (1971), 291-307.

38. Kohonen, T. An introduction to neural computing. *Neural Networks* 1 (1988), 3-16.

39. Kohonen, T. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1984.

40. Kuczewski, R. M., Myers, M. H., and Crawford, W. J. Neurocomputer workstations and processors: Approaches and applications. *Proc. IEEE First International Conference on Neural Networks,* June 1987, Vol. 3, pp. 487-500.

4 1. Kung, S. Y. Parallel architectures for artificial neural nets. *Proc. International Conference on Systolic Arrays,* 1988, pp. 163-174.

42. Lippmann, R. P. An introduction to computing with neural nets. *Comput. Architect. News* 16, 1(1988), 7-25.

43. Pomerleau, D. A., Gusciora, G. L., Touretzky, D. S., and Kung, H. T. Neural network simulation at warp speed: How we got 17 million connections per second. *Proc. International Conference on Systolic Arrays,* 1988.

44. Preparata, F. P., and Vuillemin, J. The cube-connected cycles: A versatile network for parallel computations. *Comm. ACM(* May 198 I), 300-309.

45. Reed, D. A., and Fujimoto, R. M. *Multicomputer Networks: Message-Based Parallel Processing.* MIT Press, Cambridge, MA, 1987.

46. Rumelhart, D. E., and McClelland, J. L. (Eds.). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Bradford Books/MIT Press, Cambridge, MA, 1986.

47. Seitz, C. L., Athas, W. C., Flaig, C. M., Martin, A. J., Seizovic, J., Steele, C. S., and Su, W.-K. The architecture and programming of the ametek series 2010 multicomputer. *Proc. 1988 Hypercube Conference,* Pasadena, CA, Jan. 1988.

48. Sejnowski, T. J., and Rosenberg, C. R. NETtalk: A parallel network that learns to read aloud. Tech. Rep. JHU/EECS-86/01, Johns Hopkins University, Baltimore, Jan. 1986.

49. Sequin, C. H. Doubly twisted torus networks for VLSI processor arrays. *Proc. 8th Annual Symposium on Computer Architecture,* 198 1, pp. 47 l-480.

50. Shastri, L., and Feldman, J. A. Semantic networks and neural nets. Tech. Rep. 133, Department of Computer Science, University of Rochester, Rochester, NY, June 1984.

5 1. Toborg, S., and Hwang, K. Exploring neural network and optical computing technologies. In Hwang, K., and DeGroot, D. *(Eds.). Parallel Processing for Supercomputers and Artificial Intelligence,* McGraw-Hill, New York, 1989, pp. 609-660.

52. Touretzky, D. S., and Hinton, G. E. Symbols among the neurons: Details of a connectionist inference architecture. *Proc. IJCAI,* Aug. 1985, pp. 238-243.

53. Varma, A., Ghosh, J., and Georgiou, C. Reliable design of large crosspoint switching networks. *Proc. 18th* International *Symposium on Fault-Tolerant Computing,* Tokyo, June 1988, pp. 320-327.

54. Wah, B. W. Artificial neural networks and applications. *International Conference on Parallel Processing,* Tutorial, Aug. 19, 1988.

JOYDEEP GHOSH received his Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1983. In 1988, he received a Ph.D. from the University of Southern California, wehre he was the first scholar from the School of Engineering to be awarded an "All-University Predoctoral Merit Fellowship" for four years. In the summer of 1987, he participated in a preprofessional research program with the Systems Interconnection Group, IBM T. J. Watson Research Center, Yorktown Heights. He is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Texas, Austin. His research interests include concurrent computer architectures, parallel processing, and artificial neural systems.

KAI HWANG is a professor of electrical engineering and computer science at the University of Southern California and has served as Director of the Computer Research Institute at USC. Since receiving his Ph.D. from UC Berkeley in 1972, he has for the past 20 years engaged in active research on parallel processing computers with particular interest on multiprocessors, multicomputers, vector machines, and AI-oriented computers. He is Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing* and has published over 100 scientific papers and several books in the computer area. His latest book is entitled *Parallel Processingfor Supercomputers and Artificial Intelligence* (McGraw-Hill, New York, 1989). He was elected an IEEE fellow for his contributions to digital arithmetic, computer architecture, and parallel processing.