JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 14, 319–339 (1992)

Optimal Mapping of Neural-Network Learning on Message-Passing Multicomputers*

LON-CHAN CHU AND BENJAMIN W. WAH

Center for Reliable and High Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, 1101 West Springfield Avenue, Urbana, Illinois 61801

In this paper, we study the optimal mapping of the learning process in multilayer feed-forward artificial neural networks (ANNs) on message-passing multicomputers, with an objective of minimizing the completion time of the given learning algorithm. This optimization problem is NP-hard in general and cannot be solved directly even for a small number of neurons. By observing the dominance of the computation time of a parallel neural-network learning algorithm over its communication time, we present a novel approximation algorithm for mapping large neural networks on multicomputers, given a user-specified error degree that can be tolerated in the final mapping. The target ANNs we study are learned by a static learning rule, such as the back-error-propagation learning algorithm. We study both static and dynamic mapping schemes for systems with static and dynamic background workload. Experimental results for mapping on systems with static background workload, which include a network of Sun workstations and an Intel iPSC/2 Hypercube multicomputer, are found to be very close to those predicted by analysis. Experimental results for mapping on multicomputers with dynamic background workload are obtained by simulations. © 1992 Academic Press, Inc.

1. INTRODUCTION

Artificial neural networks (ANNs) show promising potentials in artificial intelligence applications [13, 14]. However, the technologies of implementing them in hardware are not fully mature; only simple and medium-scale ANNs can be implemented in VLSI at this time [8]. The flexibility of hardware implementation of ANNs is limited because its topology cannot be changed once it is implemented in hardware. As a result, ANNs are usually designed by simulations on existing computer systems. These simulations require a large amount of computational time and can be carried out more efficiently by parallel processing.

There are two approaches to improving the turnaround time of learning in ANNs. First, we can code the original learning algorithm in an existing programming language; parallel execution is detected either by a parallelizing compiler or by user-specified annotations. Second, we can map the learning algorithm on a parallel computer, achieving the same objective but running more efficiently. We choose the second approach in this paper because a greater amount of parallelism can be exploited given the well-defined nature of the learning algorithm.

The target ANNs we are studying are multilayer networks trained by the static learning rule, e.g., back-error propagation. Larger networks with an arbitrary interconnection are not considered in this paper because the mapping of these networks is not solvable under the constraints of computer resources at the present time. The target multicomputers consist of processors with local memory and dynamic background workload.

We study both static and dynamic mapping algorithms. In the static case, we assume that the mapping is unchanged throughout the learning process. We formulate the optimal mapping problem using integer programming, with constraints on feasibility, configuration, resource, and dependence. Since the general mapping problem is NP-hard, we reduce its complexity by first partitioning the multicomputer into disjoint partitions of processors according to the ratio of communication to computation times, before the neural network is clustered and mapped on the partitions. In the dynamic case, we assume that the background workload is time varying; hence, it may be necesasry to remap the neurons as workload changes. We approximate the dynamic mapping of neurons by a sequence of static mappings based on the variance of dynamic background workload and the expected time to finish the remaining learning tasks. We take this approach because it is difficult to assess the history of background workload and predict the behavior of future workload.

Related works on this problem include parallel software simulations on multiprocessors, design of generic multicomputers for ANN simulations, and implementations of computers and VLSI chips for ANNs. Kung and colleagues mapped layered ANNs on WARP, a linear array of 10 processing cells [16]. They proposed two ap-

* This research was supported by National Science Foundation Grant MIP 88-10584, National Aeronautics and Space Administration Contract NCC 2-481, and Joint Services Electronics Program Contract N00014-90-J-1270.

proaches: network partitioning and data partitioning. The partitioning of an ANN into slices is optimal because the target machine is a ring of processors, and the neural network is assumed to be layered. In partitioning data, they used the first nine cells in WARP for performing operations in the forward production phase and the tenth cell for computing weight updates in the backward training phase; that is, one weight update was done for every nine training patterns. They assumed that weight updates were usually small and that consecutive weight updates could be neglected by running several iterations of the simulations with fixed weights before updating them. In practice, weight updates are not necessarily small and the range of updates are highly application dependent. Moreover, the scheme may not be satisfactory when a large number of cells (or processors) are concerned.

Hwang and Ghosh designed generic microcomputers suitable for ANN simulations [7, 9]. They discussed design issues with regard to the processing elements and the communication-bandwidth requirements, and proposed several guidelines for designing generic multicomputers for ANN simulations. However, they used datagram routing, which might result in unpredictable network congestion. The performance of their scheme was also dependent on the system-supported routing algorithm.

The weight-update process in a multilayer ANN can be considered a sequence of matrix–vector multiplications. By exploiting this approach, Kung and Hwang transformed ANN learning to recursive matrix operations, then to data dependency graphs, and finally to a linear systolic array with a fast interconnection network [10, 11]. Active neurons in each layer were evenly distributed to the processing cells of the systolic array, and full resource utilization was obtained in many cases. In fact, we show in this paper that their scheme was optimal when the ANN is layered and the interconnection network is fast. However, they did not consider the case when the bandwidth of links was limited and nonuniform and the processors had different computational capacity. In the latter case, active neurons may not be evenly distributed to all processing cells.

A number of other multiprocessor simulations have been reported. Researchers at Edinburgh simulated ANN learning on a transputer-based Computing Surface with 42 processors [5]. Researchers at Rochester used a 128-node BBN Butterfly multicomputer for simulating ANNs [4]. Lin *et al.* described parallel implementations of neural network computations on fine-grain mesh-connected SIMD computers [12].

This paper addresses some of the deficiencies found in previous studies which either assumed a tightly coupled system or presented a heuristic mapping algorithm for a set of heterogeneous processors. The approximation algorithm we present allows ANN simulations to be carried out on a network of heterogeneous processors with a completion time bounded from the optimal completion time. Our results are useful for designing special-purpose computers for ANN simulations and for determining the suitability of an existing system for ANN applications.

Results we present in this paper are significant extensions from our earlier work [18], in which we studied only static mappings on systems with static background workload. In this paper, we present new optimality conditions for static mappings and study algorithms for dynamic mappings.

This paper is organized into eight sections. Sections 2, 3, and 4 define the models of the ANN, the target multicomputer, and the mapping scheme, respectively. Section 5 formulates the mapping problem as an integer programming problem and presents the related constraints. Section 6 discusses the solution strategies and techniques. Decomposition and partitioning algorithms, related theorems, and a branch-and-bound search are described. Section 7 describes our experiments on multicomputers with static and dynamic background workload. These include a 16-node Intel iPSC/2 computer (with static background workload) and a bus-based network of heterogeneous Sun workstations (with both static and dynamic background workload). Finally, Section 8 draws the conclusion.

2. MODEL OF THE ARTIFICIAL NEURAL NETWORK

In this section, we describe the operations of an ANN, show its representation using a task graph, and present a formal model. Our model works for ANNs with a *static learning rule;* in particular, we focus on the back-error-propagation learning rule. A static learning rule is one whose tasks are time invariant and hence can be represented by a static task graph. We only considered multilayer ANNs; ANNs without a layer structure are first transformed into a multilayer one before being mapped.

Before we present the model, we define the convention we use in this paper for representing symbols. A symbol in this paper may have a different number of *sub*scripts, and each subscript is like a placeholder. A symbol with fewer subscripts means that it is more general than the same symbol with more subscripts. For example, $t_{p,q}$ is more general than $t_{p,q,k}$, and a property described for $t_{p,q}$ is about $t_{p,q,k}$ for every possible k. A symbol used may also have multiple *super*scripts. The first superscript denotes the meaning of the symbol, while the second one denotes the meaning of the symbol including the first superscript. For example, $t^{r,*}$ denotes the optimal t^{r} among all possible t^{r} values.

2.1. Basic Operations of ANN

An ANN can be characterized by several major components: a set of neurons, pattern of interconnection, propagation rule, activation rule, output function, and learning rule.

A neuron is the basic processing unit, which is characterized by its state, an activation function, and an output function. The activation function transforms the input signals associated with their weights and its state value to a new state value. The output function transforms the state value to an output signal.

Neurons can be classified into three types: *input neurons, hidden neurons,* and *output neurons*. Input neurons receive inputs from the external environment; output neurons send signals to the external environment; and hidden neurons are invisible to the external environment.

The learning rule specifies the mechanism of modifying the strengths of connections. The neural network can learn though incremental modifications of connection weights. In general, the modification of a connection weight is a function of four terms: (1) the state value of the destination neuron of this connection; (2) the output value of the source neuron of this connection; (3) the current connection weight; and (4) the *teaching input*, which is the expected output value of the output neuron.

A multilayer neural network can be *clustered* such that if one cluster is connected to another cluster, then all neurons in the first cluster are connected to all neurons in the second. A special case is a multilayer neural net with one cluster in each layer. Note that all neurons in a cluster are *homogeneous* in the sense that they receive the same inputs, perform the same operations, and send their outputs to the same clusters of neurons.

The operations of an ANN can be classified into two phases: *production phase* and *learning phase*. The ANN works by alternating between these two phases. In the production phase, it receives input signals from the external environment and produces output signals to the external environment. In the learning phase, it receives teaching inputs, if they are provided, and modifies the connection weights according to the learning rule.

2.2. Constrained Task Graph

An ANN can be represented by an undirected *configuration graph*. A node in the configuration graph represents a cluster of identical neurons (called *neural cluster*, or simply cluster) that perform similar operations in the production and learning phases; a link represents the communication path between two clusters. The *size* of a node is defined as the amount of computations to be performed. The *width* of a link is defined as the amount of communication between the two clusters.

In this paper, we assume that each neuron is simulated on only one processor and cannot be split across processors. The ANN simulation can be characterized by a *constrained task graph* (CTG) formed by putting two configuration graphs back to back and replacing links with arcs. The first configuration graph, with arcs pointing from input neurons to output neurons, represents operations in the forward production phase. The second configuration graph, with arcs pointing from output to input neurons, represents operations in the backward learning phase. One important constraint in the CTG is that if a task node z in the first configuration graph and another task node z' in the second configuration graph represent the same cluster, then both task nodes z and z' must be mapped to the same set of processors.

An advantage of using the task-graph representation is that we do not need to go through the details of the ANN operations. Another advantage is that the task-graph representation is independent of the learning rule, as long as the learning rule is static.

2.3. Formal Neural Network Model

Our *neural network model*, $\mathbf{M}_{\text{ANN}}$, is defined formally as $\langle \mathbf{N}_G, \mathbf{I}_G \rangle$, where $\mathbf{N}_G$ is a set of task nodes and $\mathbf{I}_G$ is a matrix indicating the interconnection of task nodes. Let Z denote the number of task nodes in the CTG. Each task node z can be represented by a ternary tuple $\langle n_z, \eta_z, \sigma_z \rangle$, where n_z is the number of neurons in this task, η_z is the amount of computation for each neuron, and σ_z is the space used for each neuron. Example 2.1 shows the specification of a three-layer ANN, with full interconnection between adjacent layers.

EXAMPLE 2.1. Consider a three-layer neural network with 500, 1000, and 200 neurons in layers 1, 2, and 3, respectively, and full interconnection between adjacent layers. The ANN can be specified as follows.

Neural Network Model $\mathbf{M}_{\text{ANN}} = \langle \mathbf{N}_G, \mathbf{I}_G \rangle$
Task Node Set $\mathbf{N}_G = \{z_1, z_2, z_3, z_1', z_2', z_3'\}$, $Z = 6$.
$n_{z_1} = n_{z_1'} = 500$, $n_{z_2} = n_{z_2'} = 1000$, $n_{z_3} = n_{z_3'} = 200$.
$\eta_{z_1} = 2.49$, $\eta_{z_2} = 2.49$, $\eta_{z_3} = 4.94$, $\eta_{z_1'} = 2.73$,
$\eta_{z_2'} = 5.31$, $\eta_{z_3'} = 10.63$.
$\sigma_{z_1} = \sigma_{z_1'} = 501$ (words), $\sigma_{z_2} = \sigma_{z_2'} = 501$ (words),
$\sigma_{z_3} = \sigma_{z_3'} = 1001$ (words).

$$\text{Interconnection Matrix } \mathbf{I}_G = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

2.4. Nonlayered ANNs

The target ANNs we study in this paper are multilayer ANNs. In general, an ANN may not be layered. To map a nonlayered ANN using our mapping algorithm, it is necessary to restructure the ANN or the CTG beforehand. The restructing methods are discussed in the following. Errors incurred due to restructuring are discussed in Section 6.3.

(a) *Restructuring Nonlayered ANNs into Multilayer ANNs*. A nonlayered ANN can be first restructured into a "multilayer" ANN, before it is modeled formally as a CTG. Finding the best restructured ANN is more difficult than the original optimal mapping problem, since the quality of restructuring cannot be determined until the optimal mapping has been found. Consequently, heuristics are used, and some errors may be incurred. Note that it may not always be possible to restructure a nonlayered ANN into a multilayer one. A nonlayered ANN, however, can always be restructured into *wavefronts* of clusters such that clusters in a wavefront are active concurrently, and neurons in the same wavefront are independent. Since a cluster in a wavefront may communicate with clusters not in its neighboring wavefronts, we call the restructured ANN with wavefronts a *semilayered ANN;* its mapping on multicomputers can be found by our proposed algorithm.

(b) *Restructuring Constrained Task Graph*. Another way to cope with a nonlayered ANN is to restructure the CTG. This is especially useful for handling *singular* task nodes in the CTG, whose size is very limited as compared to the majority of task nodes in the CTG. Since a singular task node is relatively small, it can be merged with another task node. The error incurred by restructuring can be derived based on the ratio of the size of the singular task node to the size of the larger task node into which the singular one is merged. Note that the optimal restructuring of a CTG is difficult; heuristic restructuring algorithms with a guaranteed error bound are studied in this paper.

3. MODEL OF THE MULTICOMPUTER

In this section, the basic architecture of a multicomputer and its dynamic background workload are described. We then formalize the model of the multicomputer.

3.1. Architecture of the Multicomputer

A multicomputer is a system consisting of a set of processors with local memory, a set of communication links connecting these processors, background workload descriptors, and queuing disciplines.

A processors consists of a processing unit, its background workload descriptor, local memory, and a set of communication ports through which it can communicate with other processors. The computational power of a processor is characterized by the *execution time per unit computation*, which includes processing and memory-access activities. The size of local memory of each processor is a constraint in our mapping problem.

The interconnection networks in most multicomputers can be classified into one of four classes: point-to-point, multistage interconnection networks, crossbar, and bus. The complexity of the routing problem is highly dependent on the interconnection network and the traffic on it. The routing problem for a bus is the simplest, while that for a point-to-point network is the most difficult. The latter can be simplified if a broadcast mechanism is supported. In this paper, we consider only point-to-point and bus networks. We treat multistage and crossbar networks as fully connected point-to-point networks.

The load due to traffic in the interconnection network is considered part of the background workload of processors. This simplification is appropriate since the parallel simulation of an ANN alternates between computation and communication, and communication can be included as part of the computational overhead. Hereafter, we refer to the background workload as the combined effect of the real background workload in a processor and the random traffic in the interconnection network.

A communication link i can be modeled by a setup time τ_i^s, a transfer rate r_i, and a set $\mathbf{P}_i^{\text{supp}}$ of processors it supports. The communication setup time includes the setup time of the physical link and the overhead for processing the setup. This parameter can be obtained for a real system by measuring the time for sending a null message. The transfer rate of a link indicates the number of unit data that can be transmitted via this link per unit time. The overhead for processing transmission is included in computing the transmission time. The transmission time τ_i^t per unit datum is the reciprocal of the transfer rate. The processors supported by a link are those that communicate with other processors via this link.

Note that overlapped computations and communications are allowed. If the associated overhead is small, then such overlap should be exploited in the mapping.

3.2. Dynamic Workload

In general, several processes may be active in a processor during the simulation, and the dynamic workload may affect the optimal mapping. This effect was not considered in our earlier work [18] and is included here.

In this paper, the background workload is defined as the reciprocal of the *processor utilization*, which measures the *percentage* of computational power allocated to the ANN simulation; that is, a high background workload means a low processor utilization. Specifically, the workload $\omega_{i,q}$ in processor i and time quantum q is $1/\mu_{i,q}$, where $\mu_{i,q}$ is the associated processor utilization. Note that $0 < \mu_{i,q} \leq 1$ and $\omega_{i,q} \geq 1$, and that the time quantum is large enough to allow $\mu_{i,q}$ to be nonzero. Without loss of generality, the time quantum is set to be the time for one iteration of the parallel ANN simulation.

In a degenerate case, ω_i is 1 for every processor i. This is the single-user case considered in our earlier work [18]. Another interesting case is one in which the workload is highly unbalanced; an example is a server–client organization. Given the information on background workload, the distribution of neurons will be inversely related to the workload of processors.

3.3. Formal Model of the Multicomputer

A *multicomputer model* $\mathbf{M_M}$ is defined formally as $\langle \mathbf{MC}, \mathbf{PS}, \mathbf{LS} \rangle$, where **MC** is the multicomputer configuration, **PS** is the processor specification, and **LS** is the link specification.

The *multicomputer configuration* **MC** is a 5-ary tuple $\langle \mathbf{P}, \mathbf{I_M}, \mathbf{L}, \mathbf{P_I}, \mathbf{P_O} \rangle$, where **P** is the set of processors, $\mathbf{I_M}$ is the interconnection matrix specifying the interconnection of processors, **L** is the set of links, $\mathbf{P_I}$ is the set of processors that have input facilities, and $\mathbf{P_O}$ is the set of processors that have output facilities.

The *processor specification* **PS** is a 5-ary tuple $\langle \tau^c, m, \kappa^{ol}, \tau^o, \mathbf{WL} \rangle$, where τ^c is the execution time per unit computation, m is the size of the local memory, κ^{ol} is a binary variable specifying overlapped computations and communications, τ^o is the overhead of overlapped operations, and **WL** is the background workload descriptor. If overlapped computation and communication are allowed, then $\kappa^{ol} = 1$; otherwise, $\kappa^{ol} = 0$.

The synthetic background workload descriptor **WL** is a 6-ary tuple $\langle p_0, p_1, p_2, \delta, b_u, b_l \rangle$, where p_0, p_1, and p_2 are the probabilities that the background workload in the next iteration of parallel ANN simulation will remain the same, increase, or decrease, respectively; δ is the slope of change in background workload if the background workload increases or decreases, and b_u is the upper bound and b_l is the lower bound on background workload. Note that $p_0 + p_1 + p_2 = 1$. The detailed procedure of generating synthetic background workload is described in Section 7.2.

The *link specification* **LS** is a ternary tuple $\langle r, \tau^s, \mathbf{P}^{supp} \rangle$, where r is the data transfer rate of this link, τ^s is the corresponding setup time, and $\mathbf{P}^{supp}$ is the set of processors supported by this link. Note that only one processor is assumed to transmit on a link at any time. If k processors need to transmit on link l concurrently, then k *virtual* links $l_1, \ldots, l_k$ can be used instead of a single *real* link.

The above definition must be specified for each different processor and each different link in the multicomputer. Example 3.1 illustrates the model of a network of three heterogeneous Sun workstations with dynamic background workload.

EXAMPLE 3.1. Consider a network of three heterogeneous Sun workstations connected by Ethernet and having dynamic background workload.

Multicomputer Model $\mathbf{M_M} = \langle \mathbf{MC}, \mathbf{PS}, \mathbf{LS} \rangle$
  Multicomputer Configuration $\mathbf{MC} = \langle \mathbf{P}, \mathbf{I_M}, \mathbf{L}, \mathbf{P_I}, \mathbf{P_O} \rangle$
    Processor Set $\mathbf{P} = \{0, 1, 2\}$, $P = 3$.

$$\text{Interconnection Matrix } \mathbf{I_M} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

    Link Set $\mathbf{L} = \{0\}$, $L = |\mathbf{L}| = 1$. /* Ethernet Bus */
    Input Processor Set $\mathbf{P_I} = \{0, 1, 2\}$
    Output Processor Set $\mathbf{P_O} = \{0, 1, 2\}$
  Processor Specification $\mathbf{PS} = \langle \tau^c, m, \kappa^{ol}, \tau^o, \mathbf{WL} \rangle$
    Execution Time Per Unit Computation τ^c:
      $\tau_0^c = 28.5$ (ms), $\tau_1^c = 25.5$ (ms), $\tau_2^c = 16.7$ (ms).
    Local Memory Size m (max {local real memory, disk swap space}):
      $m_0 = 3$ (Mwords), $m_1 = 5$(Mwords), $m_2 = 10$(Mwords).
    Overlapping Feature $\kappa_i^{ol} = 0$, $\forall\, i$.
    Overlapping Overhead $\tau_i^o = 0$, $\forall\, i$.
    Background Workload Descriptor $\mathbf{WL} = \langle p_0, p_1, p_2, \delta, b_u, b_l \rangle$
      Machine 0: $p_0 = 0.24$, $p_1 = 0.71$, $p_2 = 0.05$, $\delta = 0.70$, $b_u = 25$, $b_l = 1$
      Machine 1: $p_0 = 0.13$, $p_1 = 0.57$, $p_2 = 0.30$, $\delta = 0.78$, $b_u = 25$, $b_l = 1$
      Machine 2: $p_0 = 0.29$, $p_1 = 0.53$, $p_2 = 0.18$, $\delta = 0.59$, $b_u = 25$, $b_l = 1$
  Link Specification $\mathbf{LS} = \langle r, \tau^s, \mathbf{P}^{supp} \rangle$
    Data Transfer Rate $r(1/\tau^t) = 0.188$ (word/μs)
    Setup Time $\tau^s = 108.36$ (ms) (one-time cost)
    Supported Processor Set $\mathbf{P}^{supp} = \{0, 1, 2\}$.

4. MAPPING SCHEME

The mapping of an ANN simulation on a multicomputer includes the assignment of neurons (or simulation tasks) to processors and the routing of data across the interconnection network. The assignment of neurons must meet constraints on integrity, feasibility, and resources. The routing scheme must meet constraints on dependency, feasibility, resources, and configurations. These constraints are specified in detail in Section 5.2.

A solution to the mapping problem is called a *mapping scheme* (Φ) and is defined formally as a 4-ary tuple $\Phi(\mathbf{M_{ANN}}, \mathbf{M_M}, \Phi_A, \Phi_R)$, where $\mathbf{M_{ANN}}$ is the ANN model, $\mathbf{M_M}$ is the multicomputer model, Φ_A is the *assignment scheme*, and Φ_R is the *routing scheme*. Φ_A and Φ_R are related since the routing of data cannot be determined until the clusters involved have been assigned to processors.

Φ_A can be represented as an integer-valued *assignment matrix* **A** of size Z-by-P. Recall that Z is the number of task nodes and P is the number of processors. The element $a_{i,j}$ of matrix **A** indicates the number of neurons in task node i assigned to processor j. When $a_{i,j} > 0$, processor j is called a *home processor* of task node i (or home processor of the corresponding neural cluster).

The computation of an ANN simulation can be broken into several *computation segments* (or simply segments) according to the dependence constraints. Each segment can start only after all its predecessors have finished. A segment is defined by its *entry* and *exit points*. An entry point of a segment at a particular processor pinpoints the time at which this processor receives the first frame from its predecessor processors, which simulate neurons that send outputs to this segment. An exit point pinpoints the time at which this processor sends the last frame to its successor processors, which simulate neurons receiving data from this segment. The overlap of computation and communication in a processor is the interval between the entry point and the *commit point*, which identifies the time at which this processor finishes receiving the last frame from its predecessor processors for the current segment.

Formally, the jth segment in processor i is denoted by $s_{i,j} = \langle s_{i,j}^{\text{entry}}, s_{i,j}^{\text{commit}}, s_{i,j}^{\text{exit}} \rangle$, where $s_{i,j}^{\text{entry}}$, $s_{i,j}^{\text{commit}}$, and $s_{i,j}^{\text{exit}}$ are the entry, commit, and exit points, respectively. Note that two processors may have different entry and exit points for the same segment. Let h denote the *height* of the ANN, which is the number of clusters encountered along the longest acyclic path from the input cluster to the output cluster. In a multilayer ANN, h is simply the number of layers. Let $\mathbf{K}_q$ denote the set of task nodes (or clusters) involved in segment q. The maximum number of segments is $2h$, since h segments are needed for the production phase, and another h for the learning phase. For the ANN in Example 2.1, h is 3.

Φ_R can be represented by a *routing vector* of cardinality L. Recall that L is the number of links. The lth element of $\mathbf{R}$ is a set of Ξ_l of 4-ary tuples that keeps the statistics of communication on link l. The 4-ary tuple is $\langle p', f_{p,q,k}, t_{p,q,k}^{s,l}, t_{p,q,k}^{u,l} \rangle$, where p' denotes the processor that transmits frames via link l, $f_{p,q,k}$ denotes the data frame produced in processor k and used in computation segment q in processor p, $t_{p,q,k}^{s,l}$ denotes the start time for transmission using link l, and $t_{p,q,k}^{u,l}$ denotes the time period for using link l. The component of the routing vector is a set rather than a number, because a link can be used at different times by different processors and all usage must be described.

EXAMPLE 4.1. Consider the ANN presented in Example 2.1 and the multicomputer in Example 3.1. Part of a possible assignment matrix $\mathbf{A}$ showing the mapping of neurons to processors is

$$\begin{bmatrix} 96 & 157 & 247 \\ 195 & 312 & 493 \\ 39 & 62 & 99 \end{bmatrix}.$$

One possible 4-ary member of the set Ξ_0 is $\langle p' = 0, f_{1,2,0}, t_{1,2,0}^{s,0} = 26198\ (\mu\text{s}), t_{1,2,0}^{u,0} = 319\ (\mu\text{s}) \rangle$. This means that processor 0 transmits the frame it produces for computation segment 2 in processor 1, and the transmission via link 0 starts at time 26198 μs and takes 319 μs.

5. PROBLEM FORMULATION

The objective of the mapping problem is the minimization of the completion time for training the ANN, which is a function of computation and communication times. The computation time includes the time for computing neuron outputs in the production phase and the time for updating weights in the learning phase. The communication time includes the time for sending neuron output signals to the successor processors in the production phase and the time for sending error signals to the successor processors in the learning phase.

5.1. Mathematical Formulation

The optimal mapping problem can be formulated mathematically as follows. Given a neural network model $\mathbf{M}_{\text{ANN}}$ and a multicomputer model $\mathbf{M}_{\text{M}}$, find an assignment matrix $\mathbf{A}$ and a routing vector $\mathbf{R}$ to achieve the optimal objective function **OBJ.**

$$\mathbf{OBJ} = \min_{\mathbf{A},\mathbf{R}} \max_{p \in \mathbf{P}} T_p^{\text{EXEC}}(\mathbf{A}, \mathbf{R}), \tag{5.1}$$

such that $\mathbf{A}$ and $\mathbf{R}$ satisfy the constraints on feasibility, configurations, resources, and dependence. A maximization is carried out to find the completion time that is determined by the last processor that finishes the simulation. For brevity, T_p^{EXEC} is used instead of $T_p^{\text{EXEC}}(\mathbf{A}, \mathbf{R})$ in the following discussion.

T_p^{EXEC} of a processor can be formulated as the sum of the times $T_{p,q}^{\text{EXEC}}$ for segment q in processor p. Since there are $2h$ segments,

$$T_p^{\text{EXEC}} = \sum_{q=1}^{2h} T_{p,q}^{\text{EXEC}}. \tag{5.2}$$

$T_{p,q}^{\text{EXEC}}$ should include the computation time $T_{p,q}^{\text{COMP}}$ and the communication time $T_{p,q}^{\text{COMM}}$. The computation time of a segment is the sum of execution times of simulations for neurons corresponding to it:

$$T_{p,q}^{\text{COMP}} = \tau_p^{c} \sum_{z \in \mathbf{K}_q} a_{z,p}\, \eta_z. \tag{5.3}$$

Recall that $\mathbf{K}_q$ is the set of task nodes (or clusters) involved in segment q. The communication time $T_{p,q}^{\text{COMM}}$ is the time interval from the previous exit point to the current commit point:

$$T_{p,q}^{\text{COMM}} = t_{p,q}^{\text{commit}} - t_{p,q-1}^{\text{exit}}. \tag{5.4}$$

Communication can be overlapped with computation after the first frame arrives. The idle time between the previous exit time and the time at which the first frame arrives is called the *bubble time* and is denoted by $T_{p,q}^{\text{BUBBLE}}$. Note that $T_{p,q}^{\text{BUBBLE}} \leq T_{p,q}^{\text{COMM}}$. The execution time for segment q in processor p can be written as

$$T_{p,q}^{\text{EXEC}} = \kappa_p^{\text{ol}}\, T_{p,q}^{\text{BUBBLE}} + (1 - \kappa_p^{\text{ol}})\, T_{p,q}^{\text{COMM}} + T_{p,q}^{\text{COMP}}. \tag{5.5}$$

By substituting Eqs. (5.5) and (5.2) into Eq. (5.1), the objective function **OBJ** can be rewritten as

$$\mathbf{OBJ} = \min_{\mathbf{A,R}} \max_{p \in \mathbf{P}} \sum_{q=1}^{2h} (\kappa_p^{\text{ol}}\, T_{p,q}^{\text{BUBBLE}} + (1 - \kappa_p^{\text{ol}})\, T_{p,q}^{\text{COMM}} + T_{p,q}^{\text{COMP}}). \tag{5.6}$$

The bubble time $T_{p,q}^{\text{BUBBLE}}$ is equal to the time interval between the exit point of segment $q - 1$ and the time at which the first useful frame arrives. To determine the arrival time of the first frame, the arrival time of each frame must be known. Let $t_{p,q,k}^{\text{arrival}}$ be the arrival time of the frame containing values produced in processor k for segment q in processor p. If $k = p$, then $t_{p,q,k}^{\text{arrival}}$ is set to $t_{p,q-1}^{\text{exit}}$. Hence, $T_{p,q}^{\text{BUBBLE}}$ is

$$T_{p,q}^{\text{BUBBLE}} \triangleq \min_{k \in \mathbf{P}}(t_{p,q,k}^{\text{arrival}} - t_{p,q-1}^{\text{exit}}). \tag{5.7}$$

The arrival time of each frame depends on the exit point of the previous segment at the source processor and the traffic along the communication path. It can be formulated as

$$t_{p,q,k}^{\text{arrival}} \triangleq t_{k,q-1}^{\text{exit}} + T^{\text{PATH}}(\Lambda_{p,q,k}), \tag{5.8}$$

where $T^{\text{PATH}}(\Lambda_{p,q,k})$ is the time needed to send the frame containing values produced in processor k for segment q in processor p, and $\Lambda_{p,q,k}$ denotes the path consisting of links. This represents the average transmission time along the path from processor k to processor p and the average traffic delay at the intermediate processors. This time can be written as

$$T^{\text{PATH}}(\Lambda_{p,q,k}) \triangleq \sum_{\substack{x \in \Lambda_{p,q,k} \\ y \text{ is the next link to } x \\ \text{in path } \Lambda_{p,q,k}}} (t_{p,q,k}^{\text{u},x} + T^{\text{DELAY}}(f_{p,q,k}, x, y)), \tag{5.9}$$

where $T^{\text{DELAY}}(f_{p,q,k}, x, y)$ is the delay time in the processor between links x and y. Recall that $t_{p,q,k}^{\text{u},x}$ is the link usage time and is defined by $\tau_x^{\text{s}} + \sigma_{f_{p,q,k}}\, \tau_x^{\text{t}}$, where $\sigma_{f_{p,q,k}}$ is the size of frame $f_{p,q,k}$. The delay time is highly dependent on traffic on links x and y, with frames arriving first being transmitted first. The delay function for frame $f_{p,q,k}$ can be written as

$$T^{\text{DELAY}}(f_{p,q,k}, x, y) \triangleq t_{p,q,k}^{\text{s},y} - (t_{p,q,k}^{\text{s},x} + t_{p,q,k}^{\text{u},x}). \tag{5.10}$$

The entry time $t_{p,q}^{\text{entry}}$, the commit time $t_{p,q}^{\text{commit}}$, and the exit time $t_{p,q}^{\text{exit}}$ are defined as

$$t_{p,q}^{\text{entry}} \triangleq \min_{k \in \mathbf{P}} t_{p,q,k}^{\text{arrival}}. \tag{5.11}$$

$$t_{p,q}^{\text{commit}} \triangleq \max_{k \in \mathbf{P}} t_{p,q,k}^{\text{arrival}}. \tag{5.12}$$

$$t_{p,q}^{\text{exit}} \triangleq t_{p,q-1}^{\text{exit}} + T_{p,q}^{\text{EXEC}}. \tag{5.13}$$

The link start times $t_{p,q,k}^{\text{s},x}$ and $t_{p,q,k}^{\text{s},y}$ and the link usage time $t_{p,q,k}^{\text{u},x}$ in Eq. (5.10) are specified in the mapping scheme.

In summary, the objective function **OBJ** for a mapping scheme can be completely determined by combining Eqs. (5.6) through (5.13).

As an example, the objective function for mapping the neural network in Example 2.1 on the multicomputer in Example 3.1 is

$$\mathbf{OBJ} = \min_{\mathbf{A,R}} \max_{p \in \mathbf{P}} \sum_{q=1}^{2h} (T_{p,q}^{\text{COMM}} + T_{p,q}^{\text{COMP}}). \tag{5.14}$$

When a mapping scheme is determined, $a_{z,p}$, $t_{p,q}^{\text{commit}}$, and $t_{p,q}^{\text{exit}}$ can be determined accordingly. Then, by using Eqs. (5.3) and (5.4) the above objective function can be calculated.

5.2. Constraints

Four groups of constraints must be satisfied: feasibility, configuration, resource, and dependence.

The feasibility constraints include the feasibility of assignment (Eq. (5.15a)) and the feasibility of link allocation (Eq. (5.15b)). The feasibility of assignment requires that all neurons in each cluster be assigned to a subset of processors and each neuron be assigned to exactly one processor. This constraint is involved when assigning the neurons. Note that the feasibility of assignment is checked only when the production task nodes are mapped, since the assignments of the learning task nodes are constrained to be the same as those for the production task nodes. The feasibility of link allocation requires that a communication link not be allocated again during the period when it is used. This constraint is involved when mapping communication.

Feasibility Constraints

$$\sum_{i=0}^{P-1} a_{z,i} = n_z \quad \forall\, z = 1, \ldots, Z/2 \tag{5.15a}$$

$$t^{s,i}_{p,q,k} \notin [t^{s,i}_{p',q',k'},\ t^{s,i}_{p',q',k'} + t^{u,i}_{p',q',k'}]$$
$$\forall\ p' \neq p,\ q' \neq q,\ k' \neq k,$$
$$\forall\ i = 0, \ldots, L - 1. \qquad (5.15b)$$

The configuration constraints include constraint on the transmitting processor (Eq. (5.16)). This constraint requires that the processor transmitting a data frame via a link be a member of the set of processors supported by this link. This constraint is involved when the transmitting processor is granted.

Configuration Constraints

$$p' \in \mathbf{P}^{\text{supp}}_i, \quad \forall\ \langle p', f_{p,q,k}, t^{s,i}_{p,q,k}, t^{u,i}_{p,q,k}\rangle \in \Xi_i,$$
$$\forall\ i = 0, \ldots, L - 1. \qquad (5.16)$$

The resource constraint is the constraint on local memory (Eq. (5.17)), which requires that the total amount of space allocated for computation in a processor not exceed the limit of its local memory.

Resource Constraints

$$\sum_{z=1}^{Z/2} a_{z,i}\, \sigma_z \leq m_i, \quad \forall\ i = 0, \ldots, P - 1. \qquad (5.17)$$

The dependence constraints include the usage dependence (Eq. (5.18a)) and the production dependence (Eq. (5.18b)). The usage dependence requires that an output value of a neuron be used only after its value has been produced. This constraint is involved when the neuron output value at its home processor is to be transmitted to other processors. The production dependence requires that an output value be produced only after all its required input data arrive. This constraint is involved when an output value is to be produced.

Dependence Constraints

$$t_o(\pi_p(z, p) = 1) < t_o(\pi_u(z, p) = 1),$$
$$\text{if } a_{z,p} > 0,\ \forall\ p = 0, \ldots, P - 1,\ \forall\ z = 1, \ldots, Z, \quad (5.18a)$$

$$t_o(\pi_a(z, p) = 1) < t_o(\pi_u(z, p) = 1),$$
$$\forall\ p = 0, \ldots, P - 1,\ \forall\ z = 1, \ldots, Z, \quad (5.18b)$$

where π_p is the production-occurrence function, π_u is the usage occurrence function, π_a is the arrival-occurrence function, and t_o is the occurrence-time function. The occurrence function is 1 if the corresponding event occurs; otherwise, it is 0. For example, $\pi_p(z, p) = 1$ if outputs of task node z are produced in processor p, otherwise, 0. π_u and π_a are defined accordingly.

For mapping the neural network in Example 2.1 on the multicomputer in Example 3.1, constraints (5.15a) can be written as

$$a_{1,0} + a_{1,1} + a_{1,2} = 500, \quad a_{2,0} + a_{2,1} + a_{2,2} = 1000,$$
$$a_{3,0} + a_{3,1} + a_{3,2} = 200.$$

Constraint (5.17) can be written as

$$501\, a_{1,0} + 501\, a_{2,0} + 1001\, a_{3,0} \leq 3 \times 10^6,$$
$$501\, a_{1,1} + 501\, a_{2,1} + 1001\, a_{3,1} \leq 5 \times 10^6,$$
$$501\, a_{1,2} + 501\, a_{2,2} + 1001\, a_{3,2} \leq 1 \times 10^7.$$

5.3. Complexity

The integer programming formulation described in the last two sections has a nonlinear objective function as well as nonlinear constraints (Eqs. (5.15b) and (5.16)). To understand the complexity of the formulation, we first derive the number of variables it uses. These variables are due to the elements of the assignment matrix **A** and those of the routing vector **R**. $n_{\mathbf{A}}$, the number of variable items in matrix **A,** is simply equal to half of the number of elements of matrix **A** because the production and learning phases have the same assignment, i.e.,

$$n_{\mathbf{A}} = Z\,P/2. \qquad (5.19)$$

The number of elements in vector **R** is equal to its cardinality, i.e., L. However, each element Ξ_i in the routing vector **R** is itself a set of 4-ary tuples, each with three variable items, p, f, and t_s. The number of tuples in set Ξ_i is dependent on the number of routing problems ($2h$), number of processors (P), and *diameter* (D_M) of the multicomputer. For an ANN with only one cluster in each layer, each layer in the CTG is accompanied by a routing problem. Hence, there are $2h$ routing problems. The diameter of a multicomputer is the maximum length of the shortest path between any pair of processors if each link has unit length. Let $n_{\mathbf{R}}$ be the number of variable items in vector **R**. Then,

$$n_{\mathbf{R}} \leq n^{\max}_{\mathbf{R}} = 6\,h\,L\,P\,D_M. \qquad (5.20)$$

n_Φ, the number of variable items in the mapping, is equal to the number of variable items in **A** and **R**:

$$n_\Phi = n_{\mathbf{A}} + n_{\mathbf{R}} \leq n^{\max}_\Phi = \frac{Z\,P}{2} + 6\,h\,L\,P\,D_M. \qquad (5.21)$$

$n^{\max}_\Phi$ is very large in most cases; however, simplification of the mapping problem with negligible error is possible because the computation time generally dominates over the communication time. These simplification techniques are introduced in Section 6.1.

For mapping the neural network in Example 2.1 on the multicomputer in Example 3.1,

$$n_{\mathbf{A}} = 3 \times 3 = 9 \quad \text{and } n_{\mathbf{R}}^{\max} = 6 \times 2 \times 1 \times 3 \times 1 = 36.$$

For mapping the same ANN on a 16-node Hypercube computer,

$$n_{\mathbf{A}} = 3 \times 16 = 48 \quad \text{and} \quad n_{\mathbf{R}}^{\max} = 6 \times 2 \times 32 \times 16 \times 4 = 24{,}576.$$

$n_{\mathbf{R}}^{\max}$ for the 16-node Hypercube is very large, though it can be reduced dramatically through simplification techniques. For example, if the 16-node Hypercube is grouped in two partitions, then

$$n_{\mathbf{R}}^{\max,\text{simplified}} = 6 \times 2 \times 1 \times 2 \times 1 = 24.$$

The complexity of the integer programming formulation also depends on the number of possible values that each variable can acquire.

The complexity of the routing problem is illustrated as follows. Consider a case in which each processor is associated with a set of frames to be migrated, and each frame is also associated with a set of destination processors. The routing problem entails the migration of every frame from its home processor, which produces this frame, to its destination processors so that the completion time is minimized. This routing problem, called the *multiple partial broadcasting* problem, is very hard to solve for large interconnection networks because its complexity is higher than that of traditional NP-complete communication problems, such as the optimum communication spanning tree and the minimum broadcast time [6] problems.

The mapping problem formulated degenerates into the traditional precedence constrained scheduling problem if the communication overhead is neglected. The latter has been proven to be NP-complete by transformation from *3SAT* [17].

5.4. Dynamic Mapping Strategy

Since the multicomputer may have dynamic background workload, the mapping of an ANN simulation should be adjusted when the background workload changes. As it is difficult to determine the best time for migrating a cluster, we choose to remap the ANN simulation whenever the background workload changes significantly and the benefit of migration exceeds its cost.

The decision to perform remapping depends on (1) the *current simulation time* t^{sim} for one iteration of parallel ANN simulation; (2) the *predicted simulation time* $\hat{t}^{\text{sim}}$ for one iteration of ANN simulation; (3) the *expected mapping time* $\hat{t}^{\text{map}}$ for finding the optimal mapping; (4) the *remaining simulation time* t^{rem} which predicts the time for finishing the remaining simulation tasks based on the current simulation time t^{sim}; and (5) the *predicted remaining simulation time* $\hat{t}^{\text{rem}}$ which predicts the time for finishing the remaining simulation tasks based on the predicted simulation time $\hat{t}^{\text{sim}}$. By definition, $\hat{t}^{\text{rem}}$ is

$$\hat{t}^{\text{rem}} = t^{\text{rem}} \times \frac{\hat{t}^{\text{sim}}}{t^{\text{sim}}}. \tag{5.22}$$

Let t^{gain} denote the gain due to remapping. It is computed as the difference between the remaining simulation time without remapping and the total simulation time after remapping and the expected remapping time. That is,

$$t^{\text{gain}} \triangleq t^{\text{rem}} - \hat{t}^{\text{rem}} - \hat{t}^{\text{map}}. \tag{5.23}$$

Remapping should be carried out if there is a positive gain.

After the new optimal mapping is found, the data for neuron states must be migrated across different processors, and the ANN simulation resumed. In our analysis, the time for migrating data is included in the mapping time.

6. SOLUTION STRATEGIES, TECHNIQUES, AND PROPERTIES

The mapping problem can be simplified with negligible error when the computation time dominates the communication time. This dominance occurs either when the number of neurons in each cluster is large or when the communication time is relatively small. Using this dominance, we describe a formulation of the optimal mapping problem and present an approximation algorithm for finding a mapping with a guaranteed deviation from the optimal one.

6.1. Overall Strategy for Solving the Mapping Problem

The mapping problem can be simplified by observing that the computation time dominates the communication time, at least within a local subset of processors called a *partition*. Within a partition, routing can be carried out heuristically. Our strategy involves three steps.

(1) Partition the multicomputer into disjoint groups such that the ratio of the heuristic communication time of neural network simulation to the best computation time within a partition does not exceed a user-specified error degree ε. Let ε_{p} be the maximum of the ratios for all partitions. Then $\varepsilon_{\text{p}} \leq \varepsilon$. (The calculation of these ratios and the partitioning algorithm are described in Section 6.2.)

(2) Map the ANN simulation to the partitions such that the completion time of the mapping deviates from the

optimal one by an error bound ε_s which is a function of ε and ε_p. (This mapping can be found by an approximation search algorithm, and the error bound ε_s is computed in Theorem 6.5 in Section 6.3.)

(3) Find the heuristic intrapartition routing for each partition.

The mapping found includes the assignment of neural clusters to the partitions, the interpartition routing, and the heuristic intrapartition routing. The first two assignments deviate from the real optimal one (based on the given partitions) by an error bound ε_s, while the final mapping deviates from the real optimal one (based on the entire multicomputer) by the user-specified error bound ε. The guarantee of the error bound is discussed in Theorem 6.1 in Section 6.2.

Our strategy is more general than a traditional strategy that finds the optimal assignment to processors as well as the optimal routing among all processors, since our strategy degenerates to it when $\varepsilon = 0$. It is better than one that finds a mapping heuristically without an error bound. It is also more powerful than one that simply reports an error bound, because the error bound in our strategy is specified by the user, and the best mapping that deviates from the optimal one by the error bound is found.

6.2. Partitioning of the Multicomputer

In this section, the partitioning algorithm and strategies to guarantee the error bound of the mapping are described. We characterize the dominance of the computation time by the ratio of communication to computation times. The symbols we use in this section are summarized in Table 6.1 and are explained briefly below.

For a given partitioning of processors, the optimal mapping of neurons and routing of communications on these partitions can be found by a branch-and-bound search algorithm. As stated before, these two problems are tightly coupled and cannot be solved independently. $t^*_{\mathbf{Q},i}$, the computation and intrapartition communication times for cluster i in the optimal case, satisfies

$$t^*_{\mathbf{Q},i} = t^{c,*}_{\mathbf{Q},i} + t^{r,*}_{\mathbf{Q},i}. \tag{6.1}$$

Figure 6.1 illustrates the neural network, the decomposition of clusters into partitions of processors, the mapping of clusters within a partition, and the mapping of clusters on the entire multicomputer. In Fig. 6.1b, the timing diagram for the three processors in partition 1 is shown. The three blocks on the left represent the three segments of cluster 1 that are processed concurrently by the processors in partition 1. Note that $t^c_{4,1}$ includes all times during which one or more processors are performing computations for cluster 1, while $t^r_{4,1}$ represents the unoverlapped intrapartition communication times between computations in cluster 2 and cluster 4. If overlaps between communications and computations are allowed, $(t^r_{4,1} + t^R_{4,1})$ represents the minimal interval between the time at which computations of the last segment in cluster 2 are completed and the time at which the first computa-

TABLE 6.1
Summary of Symbols Used and Their Interpretations

Symbol	Meaning	Symbol	Meaning
$\Phi^{\text{part},*}$	Optimal mapping of neural clusters on the given multicomputer	Φ^{part}	Optimal assignment (without considering intrapartition communication delay), optimal interpartition routing, and heuristic intrapartition routing
$t^*_{\mathbf{Q},i}$	For $\Phi^{\text{part},*}$, time interval during which one or more processors in partition **Q** are performing computations or communications for neural cluster i, with no overlap with computations in the next neural cluster to follow	$t_{\mathbf{Q},i}$	For Φ^{part}, time interval during which one or more processors in partition **Q** are performing computations or communications for neural cluster i, with no overlap with computations in the next neural cluster to follow
$t^{c,*}_{\mathbf{Q},i}$	For $\Phi^{\text{part},*}$, time interval during which one or more processors in partition **Q** are performing computations for neural cluster i	$t^c_{\mathbf{Q},i}$	For Φ^{part}, time interval during which one or more processors in partition **Q** are performing computations for neural cluster i
$t^{r,*}_{\mathbf{Q},i}$	For $\Phi^{\text{part},*}$, time interval during which all processors in partition **Q** are performing intrapartition communications for neural cluster i, with no overlap with computations in this neural cluster or the next cluster to follow	$t^r_{\mathbf{Q},i}$	For Φ^{part}, time interval during which all processors in partition **Q** are performing intrapartition communications for neural cluster i, with no overlap with computations in this neural cluster or the next cluster to follow
$t^{R,*}_{\mathbf{Q},i}$	For $\Phi^{\text{part},*}$, same as $t^{r,*}_{\mathbf{Q},i}$ except that interpartition communication times are concerned	$t^R_{\mathbf{Q},i}$	For Φ^{part}, same as $t^r_{\mathbf{Q},i}$ except that interpartition communication times are concerned
$n_{\mathbf{Q},i}$	Number of neurons in neural cluster i assigned to partition **Q**	$\gamma_{\mathbf{Q},i}$	$= t^r_{\mathbf{Q},i}/t^c_{\mathbf{Q},i} = t^{\text{comm}}_{\mathbf{Q},i}/t^{\text{comp}}_{\mathbf{Q},i}$
$t^{\text{comp}}_{\mathbf{Q},i}$	$= t^c_{\mathbf{Q},i} \times \lvert\mathbf{Q}\rvert / n_{\mathbf{Q},i}$	$t^{\text{comm}}_{\mathbf{Q},i}$	$= t^r_{\mathbf{Q},i} \times \lvert\mathbf{Q}\rvert / n_{\mathbf{Q},i}$
$\Phi^{\text{ref},*}$	Optimal mapping of neural clusters on the reference multicomputer	$T^{\text{ref}}(\Phi)$	Completion time based on mapping Φ on the reference multicomputer
$t^*_{\mathbf{Q},i}$	$(= t^c_{\mathbf{Q},i})$ for $\Phi^{\text{ref},*}$, time interval during which virtual processor **Q** is performing computations for neural cluster i	$\underset{\sim}{t}^{R,*}_{\mathbf{Q},i}$	$(= t^R_{\mathbf{Q},i})$ for $\Phi^{\text{ref},*}$, communication times not overlapped with computation

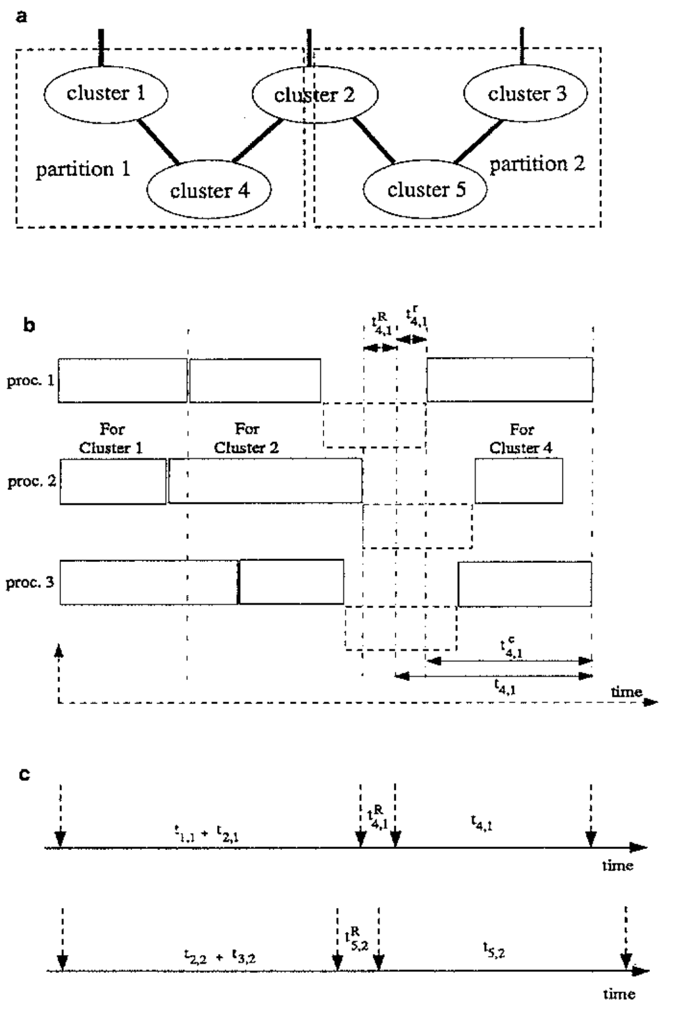

FIG. 6.1. Mapping five clusters of neurons on two partitions of processors. (a) Decomposing five clusters on two partitions of processors. (b) Timing diagram showing mapping within partition 1. (c) Timing diagram showing overall mapping.

tion in one of the segments of cluster 4 can begin. Figure 6.1c shows the time diagrams on simulating the five clusters in two partitions of processors.

Similarly, the definition of $t_{\mathbf{Q},i}$ satisfies

$$t_{\mathbf{Q},i} = t^{c}_{\mathbf{Q},i} + t^{r}_{\mathbf{Q},i} = \frac{n_{\mathbf{Q},i}}{|\mathbf{Q}|} \times (t^{\text{comp}}_{\mathbf{Q},i} + t^{\text{comm}}_{\mathbf{Q},i}). \qquad (6.2)$$

In this case, the neurons in a cluster are first allocated by ignoring their communication requirements. It is obvious that an even distribution of the neurons according to the computational power of processors in partition **Q** of processors will result in the minimal completion time $t^{c}_{\mathbf{Q},i}$ (a more general result is proved in Theorem 6.2). Note that $t^{c}_{\mathbf{Q},i}$ is a lower-bound estimate.

$t^{r}_{\mathbf{Q},i}$, the intrapartition communication time, is computed by a heuristic routing scheme. For simplicity, it is assumed that each processor broadcasts its results according to a minimum spanning tree, and that broadcasts of different processors are done sequentially. As a result, there is never any congestion involved in this communication scheme. It is, therefore, simple to compute $t^{r}_{\mathbf{Q},i}$, the interval between the time at which the last interpartition communication in cluster i is completed and the time at which the first computation in cluster i begins. Note that $t^{r}_{\mathbf{Q},i}$ is an upperbound estimate.

Another observation about the definitions in Table 6.1 is that $t^{\text{comp}}_{\mathbf{Q},i}$ is the per-neuron average computation time for cluster i, and that $t^{\text{comm}}_{\mathbf{Q},i}$ is the per-neuron average communication time for cluster i (based on a heuristic routing scheme). Since $t^{c}_{\mathbf{Q},i}$ is a lower-bound estimate and $t^{r}_{\mathbf{Q},i}$ an upper-bound estimate, $\gamma_{\mathbf{Q},i}$ represents the *worst-case* communication-to-computation-time ratio that can be experienced in partition **Q** for processing cluster i.

The last observation is that both $t^{*}_{\mathbf{Q},i}$ and $t_{\mathbf{Q},i}$ include the execution times in the production and learning phases.

In our previous work [18], we showed a method for reducing the problem complexity. Given an error bound ε of the communication-to-computation-time ratio, the multicomputer can be partitioned into several disjoint groups such that the communication-to-computation-time ratio of each group for simulating part of a given cluster is less than the bound ε. We proved that the optimal mapping on the *optimally partitioned* multicomputer with a heuristic routing scheme within each group would have a completion time no greater than $(1 + \varepsilon)$ times the completion time of the optimal mapping on the original multicomputer. The maximum of all communication-to-computation-time ratios of the partitioned multicomputer is called the *error degree*. A small error degree will result when the number of neurons in all clusters is large or when the partitions are small.

One problem with the above method is that it requires knowing the optimal partitions, knowledge that is difficult to obtain without enumeration. Moreover, finding the optimal partitioning of processors is more difficult than finding the optimal mapping itself. In this paper, we relax the above requirement on the optimal partitioning and propose an approximate mapping that is within an error bound from the optimal mapping. This is achieved by Mapping Heuristic 6.1 and Heuristic Partitioning Algorithm 6.2.

Mapping Heuristic 6.1

Neurons within a cluster can be mapped by a branch-and-bound algorithm to one or more partitions of processors with the following assumptions: (a) routing across partitions is optimal (with time $t^{R,*}_{\mathbf{Q},i}$) and (b) routing within partitions is suboptimal (with time $t^{r}_{\mathbf{Q},i}$).

Heuristic Partitioning Algorithm 6.2

(1) Select one processor not included in any partition to form a new partition. If all processors have been partitioned, then exit.

(2) For a given partition and a processor not included in any partition, if $\gamma_{\mathbf{Q},i}$ for all processors in this partition (including the newly selected processor) does not exceed the error allowance ε, then include the new processor in this partition. This step is repeated for all partitions already formed and all processors not included in any partition. Go to step 1.

To prove that the final mapping is within the error bound, a *conceptual* multicomputer, called a *reference multicomputer*, is introduced. The reference multicomputer is a multicomputer consisting of a set of *conceptual* processors, each corresponding to a partition of real processors. The reason for using a conceptual processor is that each partition is indivisible and there is no intrapartition routing. In short, there are three types of multicomputers: original multicomputer (before partitioning), partitioned multicomputer, and reference multicomputer.

The following lemmas and theorems show that the error bound can be guaranteed if Mapping Heuristic 6.1 is used.

LEMMA 6.1. *The completion time based on the optimal mapping $\Phi^{\mathrm{ref},*}$ on the reference multicomputer is no greater than the completion time based on the optimal mapping $\Phi^{\mathrm{part},*}$ on the original multicomputer. That is,*

$$T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*}) \le T(\Phi^{\mathrm{part},*}). \tag{6.3}$$

Proof. The optimal mapping $\Phi^{\mathrm{part},*}$ on the original multicomputer is also a *feasible* mapping (ignoring the corresponding intrapartition communication) on the reference multicomputer. Since there is no intrapartition communication in the reference multicomputer, we have $T^{\mathrm{ref}}(\Phi^{\mathrm{part},*}) \le T(\Phi^{\mathrm{part},*})$. By the definition of optimality of $\Phi^{\mathrm{ref},*}$, we have

$$T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*}) \le T^{\mathrm{ref}}(\Phi^{\mathrm{part},*}) \le T(\Phi^{\mathrm{part},*}). \quad \blacksquare$$

LEMMA 6.2. *The difference between $t_{\mathbf{Q},i}$ and $\underline{t}^*_{\mathbf{Q},i}$ is bounded from above by $\underline{t}^*_{\mathbf{Q},i} \times \gamma_{\mathbf{Q},i}$. That is,*

$$t_{\mathbf{Q},i} \le \underline{t}^*_{\mathbf{Q},i}(1 + \gamma_{\mathbf{Q},i}). \tag{6.4}$$

Proof. Since $\underline{t}^*_{\mathbf{Q},i}$ is the optimal completion time on the reference multicomputer, and $t^c_{\mathbf{Q},i} = \underline{t}^*_{\mathbf{Q},i}$ by definition of $t^c_{\mathbf{Q},i}$, then

$$t_{\mathbf{Q},i} = t^c_{\mathbf{Q},i} + t^r_{\mathbf{Q},i} = \underline{t}^*_{\mathbf{Q},i}\left(1 + \frac{t^r_{\mathbf{Q},i}}{t^c_{\mathbf{Q},i}}\right) \le \underline{t}^*_{\mathbf{Q},i}(1 + \gamma_{\mathbf{Q},i}). \quad \blacksquare$$

LEMMA 6.3. *Consider a multilayer ANN with* L *layers, a multicomputer with P' disjoint partitions of processors, and the corresponding reference multicomputer. Assume that every cluster i in every processor of partition $\mathbf{Q}$ has communication-to-computation-time ratio $\gamma_{\mathbf{Q},i} \triangleq t^{\mathrm{comm}}_{\mathbf{Q},i}/t^{\mathrm{comp}}_{\mathbf{Q},i}$ no greater than a predefined value ε. Let $T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*})$ and $T(\Phi^{\mathrm{part}})$ be the completion times based on Mapping Heuristic* 6.1. *Then*

$$T(\Phi^{\mathrm{part}}) \le T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*})(1 + \varepsilon). \tag{6.5}$$

Proof. Let $\mathbf{K}_l$ be the set of clusters in layer l. The completion time $T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*})$ on the reference multicomputer can be expressed as

$$T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*}) = \max_{\mathbf{Q}} \sum_{l=0}^{L-1} \sum_{i\in\mathbf{K}_l} (\underline{t}^{\mathrm{R},*}_{\mathbf{Q},i} + \underline{t}^*_{\mathbf{Q},i}).$$

The completion time $T(\Phi^{\mathrm{part}})$ on the partitioned multicomputer can be expressed as

$$T(\Phi^{\mathrm{part}}) = \max_{\mathbf{Q}} \sum_{l=0}^{L-1} \sum_{i\in\mathbf{K}_l} (\underline{t}^{\mathrm{R},*}_{\mathbf{Q},i} + t_{\mathbf{Q},i}).$$

According to Lemma 6.2, simple algebraic manipulations show that

$$\begin{aligned} T(\Phi^{\mathrm{part}}) &= \max_{\mathbf{Q}} \sum_{l=0}^{L-1} \sum_{i\in\mathbf{K}_l} (\underline{t}^{\mathrm{R},*}_{\mathbf{Q},i} + t_{\mathbf{Q},i}) \\ &\le \max_{\mathbf{Q}} \sum_{l=0}^{L-1} \sum_{i\in\mathbf{K}_l} (\underline{t}^{\mathrm{R},*}_{\mathbf{Q},i} + (1+\varepsilon)\,\underline{t}^*_{\mathbf{Q},i}) \\ &\le \max_{\mathbf{Q}} \sum_{l=0}^{L-1} \sum_{i\in\mathbf{K}_l} (1+\varepsilon)(\underline{t}^{\mathrm{R},*}_{\mathbf{Q},i} + \underline{t}^*_{\mathbf{Q},i}) \\ &\le T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*})(1+\varepsilon). \quad \blacksquare \end{aligned}$$

THEOREM 6.1. *Consider a multilayer ANN with* L *layers, and a multicomputer with* P′ *disjoint partitions of processors and its reference multicomputer. Assume that every cluster i in every processor of partition $\mathbf{Q}$ has communication-to-computation-time ratio $\gamma_{\mathbf{Q},i} \triangleq t^{\mathrm{comm}}_{\mathbf{Q},i}/t^{\mathrm{comp}}_{\mathbf{Q},i}$ no greater than a predefined value ε. Let $T(\Phi^{\mathrm{part}})$ and $T(\Phi^{\mathrm{part},*})$ be the completion times based on Mapping Heuristic* 6.1 *and on the optimal mapping, respectively. Then,*

$$T(\Phi^{\mathrm{part}}) \le T(\Phi^{\mathrm{part},*})(1 + \varepsilon). \tag{6.6}$$

Proof. By Lemmas 6.1 and 6.3, we have

$$T(\Phi^{\mathrm{part}}) \le T^{\mathrm{ref}}(\Phi^{\mathrm{ref},*})(1 + \varepsilon) \le T(\Phi^{\mathrm{part},*})(1 + \varepsilon). \quad \blacksquare$$

Theorem 6.1 guarantees that the bound on the error with respect to the optimal mapping can be achieved when Mapping Heuristic 6.1 and Heuristic Partitioning

Algorithm 6.2 are used. In our previous work [18], we were unable to guarantee this error bound. This new result is very important because it says that partitioning does not affect the error bound of the optimality of mapping, as long as it satisfies the communication-to-computation-time ratio. Therefore, any partitioning algorithm other than the optimal one will suffice.

The following theorem shows that distributing neurons within a partition according to the computational power or processors within the partition is optimal.

THEOREM 6.2. *Assume that $n_{\mathbf{Q},i}$ neurons in neural cluster i are assigned to a partition of processors* $\mathbf{Q}$ *and that the computation time dominates the communication time in partition* $\mathbf{Q}$. *The optimal assignment on* $\mathbf{Q}$ *can be obtained by distributing the $n_{\mathbf{Q},i}$ neurons evenly according to the computational power of processors. Processor j* completes at approximately $x_{i,j}\tau_j + \nu_j$, *where $x_{i,j}$ is the number of neurons in cluster i assigned to processor j. τ_j and ν_j are, respectively, the execution time per unit computation and the amount of time that processor j is not available for ANN simulation.*

Proof. Since the computation time dominates the communication time in this partition, only the computation time must be considered in the proof. Let $X_{\mathbf{Q},i}$ be a possible mapping of cluster i on $\mathbf{Q}$, and $C_{\langle X_{\mathbf{Q},i}\rangle}$ be the completion time of mapping $X_{\mathbf{Q},i}$; namely, $C_{\langle X_{\mathbf{Q},i}\rangle} = \max_{j\in\mathbf{Q}} \{x_{i,j}\tau_j + \nu_j\}$. The optimal execution time can be written as $C_{\langle X^*_{\mathbf{Q},i}\rangle} = \min_{X_{\mathbf{Q},i}} C_{\langle X_{\mathbf{Q},i}\rangle}$. Since $\sum_{j\in\mathbf{Q}} x_{i,j} = n_{\mathbf{Q},i}$, $C_{\langle X^*_{\mathbf{Q},i}\rangle}$ can be derived easily as

$$C_{\langle X^*_{\mathbf{Q},i}\rangle} = \frac{n_{\mathbf{Q},i} + \sum_{j\in\mathbf{Q}} (\nu_j/\tau_j)}{\sum_{j\in\mathbf{Q}} (1/\tau_j)}.$$

Assume that there exists another better assignment $X'_{\mathbf{Q},i}$ such that $C_{\langle X'_{\mathbf{Q},i}\rangle} = \max_{j\in\mathbf{Q}} \{x'_{i,j}\tau_j + \nu_j\} \le C_{\langle X^*_{\mathbf{Q},i}\rangle}$; then assignment $x'_{i,j}$ satisfies an inequality $x'_{i,j}\tau_j \le C_{\langle X'_{\mathbf{Q},i}\rangle} - \nu_j$. By summing all $x'_{i,j}$, we have

$$n_{\mathbf{Q},i} = \sum_{j\in\mathbf{Q}} x'_{i,j} = \sum_{j\in\mathbf{Q}} \frac{C_{\langle X'_{\mathbf{Q},i}\rangle} - \nu_j}{\tau_j} < \sum_{j\in\mathbf{Q}} \frac{C_{\langle X^*_{\mathbf{Q},i}\rangle} - \nu_j}{\tau_j} = n_{\mathbf{Q},i}.$$

A contradiction! Consequently, $C_{\langle X'_{\mathbf{Q},i}\rangle} \ge C_{\langle X^*_{\mathbf{Q},i}\rangle}$ must hold; that is, the optimal execution time is $C_{\langle X^*_{\mathbf{Q},i}\rangle}$. ■

According to Theorem 6.2, $x_{i,j}$ can be calculated by using the equality

$$x_{i,j}\tau_j = \left(\frac{n_{\mathbf{Q},i} + \sum_{j'\in\mathbf{Q}} (\nu_{j'}/\tau_{j'})}{\sum_{j'\in\mathbf{Q}} (1/\tau_{j'})} - \nu_j\right). \tag{6.7}$$

Note that if $\nu_j = 0$ for every j, then a uniform distribution according to the computational power of processors in $\mathbf{Q}$ follows from Theorem 6.2. Further, note that if

$$\frac{n_{\mathbf{Q},i} + \sum_{j'\in\mathbf{Q}} (\nu_{j'}/\tau_{j'})}{\sum_{j'\in\mathbf{Q}} (1/\tau_{j'})} < \nu_j \tag{6.8}$$

is true, then the most negative $x_{i,j}$ (Eq. (6.7)) can first be set to zero and $x_{i,k}$ can be recomputed for every $k \neq j$ in the set of processors $\mathbf{Q}$. This process may have to be repeated several times in the worst case.

COROLLARY 6.1. *In a system with homogeneous processors connected by a fast interconnection network (such as a linear systolic array assumed by Kung and Hwang* [10, 11]), *an even distribution of neurons in a cluster to all processing cells results in the minimal completion time in learning.*

Proof. Since the interconnection network is fast, the computational overhead dominates the communication overhead. According to Theorem 6.1, the entire system can be considered one partition with negligible error with respect to the optimal mapping. Further, according to Theorem 6.2, neurons should be mapped evenly to all processing elements. ■

The resource parameters of a partition q, including the set of processors $\mathbf{Q}$, can be defined as follows (refer to the definitions in Section 3.3):

$$\frac{1}{\tau^c_q} = \sum_{j\in\mathbf{Q}} \frac{1}{\tau^c_j}, \tag{6.9a}$$

$$m_q = \sum_{j\in\mathbf{Q}} m_j, \tag{6.9b}$$

$$\kappa^{ol}_q = 1 \quad \text{if} \kappa^{ol}_j = 1 \quad \text{for some } j \in \mathbf{Q}, \tag{6.9c}$$

$$\tau^o_q = \frac{\sum_{j\in\mathbf{Q}} \tau^o_j}{|\mathbf{Q}|}. \tag{6.9d}$$

After the partitions are generated, the communication links connecting one partition to another can be grouped into conceptual links in such a way that a conceptual link connecting two partitions includes all links connecting a processor in one partition to any processor in the second partition. The parameters of conceptual link λ, consisting of a set Λ of real links, can be defined as

$$\tau^s_\lambda = \frac{1}{|\Lambda|} \sum_{i\in\Lambda} \tau^s_i, \tag{6.10a}$$

$$\frac{1}{\tau^t_\lambda} = \sum_{i\in\Lambda} \frac{1}{\tau^t_i}. \tag{6.10b}$$

$\mathbf{P}^{\text{supp}}$ also needs to be modified accordingly.

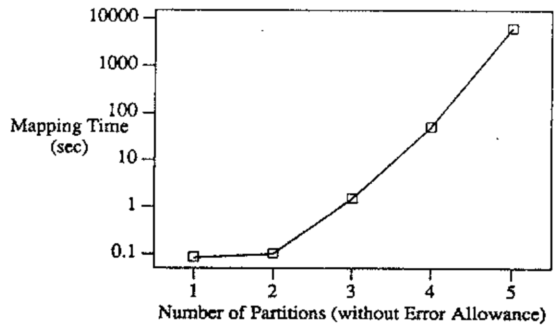

FIG. 6.2. Execution time for finding the optimal mapping of ANN FC-1 on different numbers of partitions.

The complexity of the mapping problem is dependent on the number of partitions, the interconnection of multicomputers, and the resource parameters. Figure 6.2 shows the mapping times for solving the optimal mapping of ANN FC-1 (which is described in Section 7) on different numbers of partitions. (In the special case in which there is one processor in each partition, the problem is equivalent to finding the optimal mapping of the ANN.) Note that the execution time grows exponentially with respect to the number of partitions, since the mapping problem is NP-hard. Figure 6.3 shows that the mapping times for solving the optimal mapping of fully connected ANNs with different numbers of clusters on a three-partition multicomputer. It is observed that the mapping times grow exponentially with the number of clusters.

6.3. Decomposition of Error Allowance

Our approach to solving the optimal mapping problem consists of two stages, as described in Section 6.1. Each stage can incur certain error degree in order to reduce the mapping time. The following lemma and theorems show that the total error incurred can be computed by the error degrees incurred in each stage.

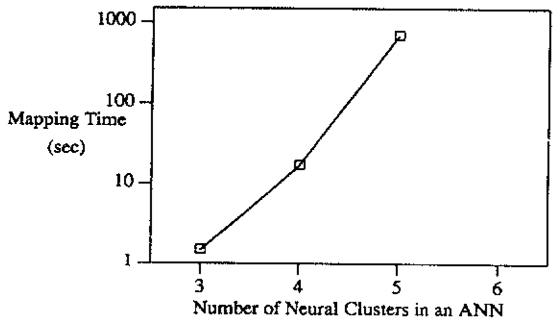

FIG. 6.3. Execution time for finding the optimal mapping of fully connected ANNs with different numbers of clusters on a three-partition multicomputer.

LEMMA 6.4. *Consider a two-stage problem solver, say S_1 and S_2, each stage incurring certain error degree, say ε_1 and ε_2. Then $\varepsilon_{1,2}$, the total error degree incurred, is bounded by*

$$\varepsilon_{1,2} \leq \varepsilon_{1,2}^{\max} = \varepsilon_1 + \varepsilon_2 + \varepsilon_1\varepsilon_2. \tag{6.11}$$

Proof. Let t^* be the optimal solution value (the completion time based on the optimal mapping in the mapping problem). Also let t_1 and t_2 be the solution values after error degrees ε_1 and ε_2, respectively, have been incurred. Then, we have

$$t_1 \leq t^*(1 + \varepsilon_1) \quad \text{and} \quad t_2 \leq t_1(1 + \varepsilon_2).$$

By combining the two equations above, we obtain

$$t_2 \leq t^*(1 + \varepsilon_1)(1 + \varepsilon_2) = t^*(1 + \varepsilon_1 + \varepsilon_2 + \varepsilon_1\varepsilon_2).$$

Therefore, the maximum of the total error degree is

$$\varepsilon_{1,2}^{\max} = \varepsilon_1 + \varepsilon_2 + \varepsilon_1\varepsilon_2.$$

The lemma is proved by taking this maximum as the upper bound. ■

THEOREM 6.3. *Consider an n-stage problem solver, say $S_1, \ldots, S_n$, each stage incurring certain error degree, say $\varepsilon_1, \ldots, \varepsilon_n$. The total error degree $\varepsilon_{1,n}$ incurred is bounded by*

$$\varepsilon_{1,n} \leq \sum_{k=1}^{n} \sum_{\forall P_k \in \Gamma_k} \prod_{i \in P_k} \varepsilon_i, \tag{6.12}$$

where P_k is a permutation $(i_1, \ldots, i_k)$ from $(1, \ldots, n)$, and Γ_k is the set of all possible permutations consisting of k items.

Proof. This theorem can be proved by applying Lemma 6.4 iteratively. First, $\varepsilon_{1,2}^{\max}$, the composite error bound incurred in stages S_1 and S_2, can be calculated by Eq. (6.11). Next, stage S_3 is included and $\varepsilon_{1,3}^{\max}$, the composite error bound based on $\varepsilon_{1,2}^{\max}$ and ε_3, is calculated using Eq. (6.11). Iteratively, $\varepsilon_{1,k}^{\max}$ can be calculated based on $\varepsilon_{1,k-1}^{\max}$ and ε_k. Finally, $\varepsilon_{1,n}^{\max}$ can be calculated. ■

The accumulation of error degrees gives the worst-case upper bound of the total error degree. For $n = 3$, the error bound is

$$\varepsilon_{1,3}^{\max} = \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_1\varepsilon_2 + \varepsilon_1\varepsilon_3 + \varepsilon_2\,\varepsilon_3 + \varepsilon_1\varepsilon_2\varepsilon_3. \tag{6.13}$$

The transformation of nonlayered ANN into an ANN with wavefronts (discussed in Section 2) can be treated as a sequence of merges, each of which is a merge of activations from the previous state. A merge may incur an error degree bounded by the communication-to-computation-time ratio. By applying the result in Theorem 6.3, the maximum total error degree incurred in the transformation can be calculated. If the error allowance is specified, we can determine whether the maximum error incurred in the transformation is within the error allowance.

The transformation of singular task nodes (discussed in Section 2.4) is also a sequence of merges. The error incurred in a merge of a given singular task node to a larger task node is bounded by the ratio of their execution times.

If the total error degree allowed is given and the error degree incurred in partitioning the multicomputer is known, then the error degree allowed in the mapping algorithm can be set based on the total error degree allowed and the error degree incurred in partitioning the multicomputer (see section 6.1). The following theorems show the error degrees allowed in this approach.

THEOREM 6.4. *Consider a problem solver consisting of two stages, say S_1 and S_2. If the total error allowed is limited to $\varepsilon_{1,2}$ and the error degree incurred in stage S_1 is ε_1, then the error degree that can be incurred in stage S_2 is bounded by*

$$\varepsilon_2 \le \varepsilon_2^{\max} = \frac{\varepsilon_{1,2} - \varepsilon_1}{1 + \varepsilon_1}. \tag{6.14}$$

Proof. To achieve the largest error allowance, let $\varepsilon_{1,2}^{\max}$ be $\varepsilon_{1,2}$ in Eq. (6.11). We have

$$\varepsilon_2^{\max} = \frac{\varepsilon_{1,2}^{\max} - \varepsilon_1}{1 + \varepsilon_1} = \frac{\varepsilon_{1,2} - \varepsilon_1}{1 + \varepsilon_1}.$$

The theorem is provided by taking $\varepsilon_2^{\max}$ as the upper bound. ■

The decomposition of errors in the n-stage problem can also be done by rearranging the terms in Eq. (6.12). By determining the error degree allowed in each stage so that each can be solved as an independent problem, the complexity of the mapping problem is reduced significantly.

THEOREM 6.5. *If the error allowed in the mapping problem is ε and the error degree (by ignoring communication time) incurred in partitioning the multicomputer is ε_p, then ε_s, the error degree allowed in mapping the neurons to the partitions, is*

$$\varepsilon_s = \frac{\varepsilon - \varepsilon_p}{1 + \varepsilon_p}. \tag{6.15}$$

Proof. Since we solve the optimal mapping problem in two stages, the result in Theorem 6.4 can be applied. By substituting $\varepsilon = \varepsilon_{1,2}$, $\varepsilon_p = \varepsilon_1$, and $\varepsilon_s = \varepsilon_2$ into Eq. (6.14) and assuming that ε_s is the worst-case error allowance, the theorem is proved. ■

6.4. Branch-and-Bound Search

The mapping problem formulated by nonlinear integer programming can be solved by a branch-and-bound search. During the search, each node represents either an assignment of a cluster to a partition or the choice of a route between two layers. One important feature of this representation is that the search branches on neural clusters instead of individual neurons. Each node is associated with a lower-bound and an upper-bound completion time. A node can be pruned if its lower bound is larger than the least upper bound. The upper-bound completion time can be obtained by a greedy search. Detailed formulations of the lower and upper bounds are described elsewhere [2] and are not shown here due to space limitations.

When the size of an ANN grows, the computation time will become more dominant over the communication time. This phenomenon is illustrated by the following example. Consider a simple multilayer ANN with L layers and N neurons in each layer. Assume the number of processors in the target multicomputer to be P. The computation time for a neuron is $O(N)$, resulting in a total computation time for a layer of $O(N^2)$. However, the communication in each layer requires N neuron outputs to be sent to a maximum of P processors. Hence, the total communication time for a layer is $O(PN)$, and the computation time is substantially larger than the communication time. This phenomenon is significant since larger ANN imply relatively small overhead on communication.

7. EXPERIMENTAL RESULTS

In this section, experimental results on multicomputers with both static and dynamic background workload are shown. Cases with static workload include a set of three heterogeneous Sun workstations connected by an Ethernet, and an Intel iPSC/2 Hypercube computer with 16-node, 8-node, and 4-node configurations. Cases with dynamic workload studied include the network of Sun workstations described above, and a multicomputer with 10 processors, 25 processors, and 100 processors, connected by either high-speed or low-speed communication links. Note that our experiments on dynamic background workload do not include the iPSC/2 Hypercube computer because it runs in a single-user mode.

We implement a program called *NeuMap* that partitions the multicomputer and solves the optimal mapping problem. We implement another program called *Dsim* that simulated multicomputers with dynamics back-

TABLE 7.1
Summary of Important Symbols Used in Section 7

Symbol	Meaning	Symbol	Meaning
T^{seq}	Completion time of one iteration of sequential ANN simulation on the fastest processor in the physical multicomputer	$T^{\parallel}$	Completion time of one iteration of parallel ANN simulation on the physical multicomputer, with user-specified approximation degree
T^{pred}	Completion time of one iteration of parallel ANN simulation predicted by NeuMap, with user-specified approximation degree	T^{sim}	Completion time of one iteration of parallel ANN simulation on the Dsim simulator, with user-specified approximation degree
Y	Speedup of parallel ANN simulation to sequential ANN simulation ($Y \triangleq T^{\mathrm{seq}}/T^{\parallel}$)	$\varepsilon^{\mathrm{user}}$	Error allowance specified by the user such that the completion time of the mapping found by NeuMap will not deviate from that of the optimal mapping by this error
$\varepsilon^{\mathrm{pred}}$	Deviation between $T^{\parallel}$ and T^{pred} ($\varepsilon^{\mathrm{pred}} \triangleq \lvert T^{\mathrm{pred}} - T^{\parallel}\rvert/T^{\parallel}$)	$\varepsilon^{\mathrm{sim}}$	Deviation between $T^{\parallel}$ and T^{sim} ($\varepsilon^{\mathrm{sim}} \triangleq \lvert T^{\mathrm{sim}} - T^{\parallel}\rvert/T^{\parallel}$)
T^{static}	Completion time of N^{iter} iterations of ANN simulation on Dsim plus the time for excecuting NeuMap once in static mapping	T^{dyn}	Completion time of N^{iter} iterations of ANN simulation on Dsim plus all the times for executing NeuMap in dynamic mapping
g	Gain in using dynamic mapping strategy over static mapping strategy ($g \triangleq T^{\mathrm{static}}/T^{\mathrm{dyn}}$)	n_k	Number of neurons in neural cluster k
$Y^{\max}$	Maximum possible speedup of parallel processing of ANN simulations (equal to the number of processors if all processors are homogeneous)	$g^{\max}$	Maximum possible gain in using dynamic mapping strategy over static one (obtained by performing remapping in every iteration of ANN simulation and ignoring the mapping overhead)
$\hat{Y}$	Speedup efficiency ($\hat{Y} \triangleq Y/Y^{\max}$)	$\hat{g}$	Gain efficiency ($\hat{g} \triangleq g/g^{\max}$)
$\mathbf{prec}_k$	Set of predecessor neural clusters of neural cluster k	$\mathbf{succ}_k$	Set of successor neural clusters of neural cluster k
η_{z_k}	Amount of computation per neuron for task node z_k, which represents neural cluster k in the production phase	$\eta_{z'_k}$	Amount of computation per neuron for task node z'_k, which represents neural cluster k in the learning phase

ground workload. Dsim allows communicaton on point-to-point links as well as broadcast buses. During dynamic mapping, Dsim is the master process: whenever Dsim decides to remap the neurons (based on the rules described in Section 5.4), Dism calls NeuMap and waits for a new mapping before proceeding with the parallel simulation. We implement Dsim as a simulator because it is difficult to reproduce a variety of dynamic background workloads on a physical computer.

Table 7.1 shows the important symbols used in this section for describing our experimental results. Table 7.2 shows the parameters of the ANN benchmarks used in our experiments. These parameters are measured with respect to the computational power of one of the three Sun workstations described in Example 3.1. For the iPSC/2 Hypercube computer, due to memory limitations, the number of neurons in each cluster is reduced to half. For multicomputers with dynamic background workload, the number of neuros in each cluster is extended by 10 times that listed in Table 7.2.

Table 7.3 shows the communication parameters used, including those for the Sun workstations and the iPSC/2 computer. The communication setup time is obtained by measuring the transmission time for a null frame, whereas the transmission time per word is obtained by applying a linear approximation over communication times for different frame sizes. Note that all communication parameters include preprocessing and postprocessing times.

7.1. Experiments on Multicomputers with Static Background Workload

When the background workload is static, it means that the workload is either time invariant or changing very slowly. This is the case in the Hypercube computer or in the network of Sun workstations running in a single-user mode. Note that all experiments are measured with respect to one iteration of the parallel ANN simulation, as all iterations are identical in processing time.

7.1.1. Experiments on Workstations with Static Background Workload

The target multicomputer is a network of three heterogeneous Sun workstations specified in Example 3.1 in Section 3.3. Machine 1 has the lowest computational power, and machine 3 has the highest. Machine 3 is used as the base machine to calculate the amount of computation in each cluster listed in Table 7.2. Each processor is assumed using virtual-circuit communication with a one-time setup cost. Broadcasts on Ethernets using datagrams are not used in our experiments due to the small number of processors, though it will be useful when the number of processors is large.

TABLE 7.2
Summary of ANN Benchmarks Used in Our Experiments

Cluster k	**prec**$_k$	**succ**$_k$	n_k	η_{zk}	η'_{zk}	n_k	η_{zk}	η'_{zk}	n_k	η_{zk}	η'_{zk}
			Fully connected multilayer feed-forward ANNs								
ANN topology			FC-1			FC-2			FC-3		
1	IN	2	500	2.49	2.73	600	2.99	3.28	200	1.06	1.08
2	1	3	1000	2.49	5.31	200	2.99	6.41	1500	1.06	2.15
3	2	OUT	200	4.94	10.63	500	1.05	2.15	200	7.35	15.98
			Hybrid multilayer feed-forward ANNs								
ANN topology			ML-1			ML-2			ML-3		
1	IN	2, 3, 4	500	2.53	2.80	200	1.05	1.14	800	4.26	4.60
2	1	5	200	2.53	5.52	300	1.05	2.17	300	4.26	8.80
3	1	5, 6	500	2.53	5.52	500	1.05	2.17	200	4.26	8.80
4	1	6	300	2.53	5.52	200	1.05	2.17	400	4.26	8.80
5	2, 3	7	400	3.56	7.58	600	3.97	8.55	200	2.73	5.34
6	3, 4	7	600	4.13	8.56	400	3.47	7.46	500	3.19	6.43
7	5, 6	OUT	200	5.08	10.69	200	4.94	10.65	400	3.74	7.51
			Nonlayered feed-forward ANNs								
ANN topology			NL-1			NL-2			NL-3		
1	IN	2, 3, 6	300	1.65	1.73	800	4.33	4.95	400	2.01	2.14
2	1	4, 5	800	1.65	3.36	500	4.33	9.02	300	2.01	4.35
3	1	8	600	1.65	3.29	400	4.33	8.75	200	2.01	4.30
4	2	8	500	4.27	8.97	400	2.76	5.46	200	1.52	3.18
5	2	7	700	4.27	8.90	800	2.76	5.39	400	1.52	3.26
6	1	8	400	1.65	3.29	400	4.35	8.75	500	2.01	4.30
7	5	8	600	3.76	7.63	500	4.35	8.51	300	2.02	4.31
8	3, 4, 6, 7	OUT	300	11.32	22.50	200	8.89	18.10	400	5.89	12.86

The predicted and experimental results are shown in Table 7.4. The experiments are conducted assuming user-specified error allowances $\varepsilon^{\text{user}} = 0$ for FC-1, FC-2, and FC-3, and $\varepsilon^{\text{user}} = 1\%$ for other ANNs. Note that the predicted error $\varepsilon^{\text{pred}}$ is around 1–2% larger than $\varepsilon^{\text{user}}$, because we did not account for overheads in synchronization, problem-partitioning, and page faults. ε^{sim}, the error incurred in simulation using Dsim, is also found to be very small.

It is observed in Table 7.4 that the amount of communication required has little effect on the speedup efficiency: $\hat{Y}$ is slightly higher for fully connected ANNs, which have a lower communication requirement.

7.1.2. *Experiments on Hypercube Computers*

The 16-node iPSC/2 Hypercube computer [1, 3] can be configured as 16-node, 8-node, and 4-node configurations. It provides virtual cut-through for interprocessor communication [15], allowing the network server to route frames by concurrent asynchronous broadcasts. Note that the broadcast parameters in Table 7.3 are measured under the condition that all processors broadcast concurrently and asynchronously rather than one processor broadcasting at a time.

The predicted and experimental results in simulating the nine ANNs listed in Table 7.2 are summarized in

TABLE 7.3
Summary of Communication Parameters Including Preprocessing and Postprocessing Times

	Workstation	Hypercube computer			
Communication parameter	Bus	Node-to-node link	16-node broadcast	8-node broadcast	4-node broadcast
τ^{s} (ms)	108.36	0.65	6.5	3.6	2.0
τ^{t} (μs)	5.33	3.95	103	48	21

TABLE 7.4
Summary of Predicted and Experimental Results in Simulating Nine ANNs Listed in Table 7.2 on Three Sun Workstations

ANN	T^{pred} (s)	$T^{\parallel}$ (s)	$\varepsilon^{\text{pred}}$ (%)	T^{seq} (s)	Y	$\hat{Y}$	T^{sim} (s)	ε^{sim} (%)
FC-1	100.95	102.22	1.25	225.47	2.23	0.995	101.10	1.09
FC-2	54.03	54.83	1.47	121.23	2.21	0.986	54.33	0.92
FC-3	74.15	74.95	1.07	167.88	2.23	0.995	74.20	1.01
ML-1	193.31	195.42	1.07	435.80	2.23	0.995	193.96	0.74
ML-2	139.06	141.02	1.39	310.40	2.20	0.982	139.52	1.06
ML-3	221.83	225.63	1.71	485.18	2.15	0.960	222.61	1.34
NL-1	318.82	319.67	0.26	704.88	2.21	0.987	320.26	0.18
NL-2	344.36	355.88	3.24	748.07	2.10	0.938	346.91	2.52
NL-3	150.81	154.87	2.62	338.93	2.19	0.978	151.84	1.95

Table 7.5. The experiments are conducted assuming user-specified error allowance $\varepsilon^{\text{user}} = 1\%$ for the 4-node cube and $\varepsilon^{\text{user}} = 2\%$ for the 8-node and 16-node cubes. As in the case with workstations, the predicted error $\varepsilon^{\text{prod}}$ is around 1–3% larger than the user-specified error allowance $\varepsilon^{\text{user}}$. This happens because synchronization and problem-partitioning overheads are not included in our model.

It is interesting to observe that the speedup efficiency is higher for smaller cubes. This occurs because in a larger cube each node has less computation, resulting in more dominance of the synchronization and problem-partitioning overheads in the performance. Further, larger cubes have higher overheads in interprocessor communication. It is also interesting to note that for the same cube, the speedup changes slowly with respect to the communication requirements, as in the workstation case.

A major limitation in using the Hypercube for ANN simulations is its limited memory space in each processor. The system lacks a virtual-memory facility from each processor to the common secondary memory, and all accesses to the secondary memory must be handled by the Cube Manager. When the number of neurons mapped to each processor is larger than the local memory capacity, part of the data must be kept in the Cube Manager. This results in high traffic between the Cube Manager and the rest of the system.

7.2. Experiments on Multicomputers with Dynamic Background Workload

When the background workload is dynamic, it means that it changes with time, and sometimes the changes may be large or fast. The static (one-time) mapping of the ANN simulation cannot adapt to the dynamic workload; the ANN simulations should be remapped if the background workload changes significantly.

The parallel ANN simulation is performed on Dsim using synthetic background workload instead of being carried out on a real multicomputer as in the previous two experiments. The major reason is that it is very difficult to reproduce a real background workload on a physical computer in order to test alternative mappings and to determine the effects of background workload on the ANN simulations.

TABLE 7.5
Summary of Predicted and Experimental Results in Simulating the Nine ANNs Listed in Table 7.2 on the iPSC/2 Hypercube Computer of Different Sizes

	4-node hypercube				8-node hypercube				16-node hypercube			
ANN	T^{pred} (s)	$T^{\parallel}$ (s)	$\varepsilon^{\text{pred}}$ (%)	Y	T^{pred} (s)	$T^{\parallel}$ (s)	$\varepsilon^{\text{pred}}$ (%)	Y	T^{pred} (s)	$T^{\parallel}$ (s)	$\varepsilon^{\text{pred}}$ (%)	Y
FC-1	2.239	2.248	0.40	3.89	1.154	1.170	1.37	7.48	0.645	0.645	0.00	13.57
FC-2	1.177	1.180	0.25	3.91	0.612	0.627	2.39	7.37	0.354	0.355	0.28	13.01
FC-3	1.672	1.667	0.30	3.87	0.880	0.880	0.00	7.34	0.528	0.524	0.76	12.32
ML-1	4.273	4.300	0.63	3.64	2.244	2.302	2.52	6.80	1.329	1.359	2.21	11.52
ML-2	3.154	3.185	0.97	3.60	1.684	1.701	1.00	6.73	1.051	1.033	1.74	11.09
ML-3	4.727	4.799	1.50	3.65	2.465	2.556	3.67	6.86	1.428	1.495	4.48	11.72
NL-1	6.797	6.848	0.74	3.68	3.540	3.585	1.26	7.03	2.044	2.031	0.64	12.41
NL-2	7.228	7.311	1.14	3.67	3.736	3.784	1.27	7.09	2.103	2.150	2.19	12.48
NL-3	3.369	3.382	0.38	3.64	1.798	1.816	0.99	6.78	1.121	1.153	2.78	10.67

In generating the synthetic background workload, each machine is associated with a background workload descriptor that is defined in Section 3.3 as a 6-ary tuple $\mathbf{WL} = \langle p_0, p_1, p_2, \delta, b_u, b_l \rangle$, where p_0, p_1, and p_2 are the probabilities that the background workload in the next iteration of parallel ANN simulation will remain the same, increase, and decrease, respectively; δ is the slope of change in background workload if the background workload increases or decreases; and b_u and b_l are the upper and lower bounds on background workload, respectively. Note that $p_0 + p_1 + p_2 = 1$. Given the background workload descriptor for each machine, after the kth iteration of ANN simulation, the background workload ω for the $(k + 1)$th iteration is generated based on the descriptor and ω in the kth iteration. The procedure for generating a synthetic background workload is summarized in the algorithm below.

Heuristic Workload Generation Algorithm 7.1

Generate a random variable from a uniform distribution $\nu \in [0, 1]$.

```
if ($\omega_k$ is not *saturated*, i.e., it is neither equal to $b_u$ nor $b_l$) then
    begin
    if ($\nu < p_0$) then
        $\omega_{k+1} := \omega_k$
    else if ($\nu < p_0 + p_1$) then
        $\omega_{k+1} := \omega_k + \delta$
    else if ($\nu \geq p_0 + p_1$) then
        $\omega_{k+1} := \omega_k - \delta$;
    if ($\omega_{k+1} \geq b_u$) then
        $\omega_{k+1} := b_u$
    else if ($\omega_{k+1} \leq b_l$) then
        $\omega_{k+1} = b_l$
    end.
if ($\omega_k$ is saturated to the upper bound, i.e., $\omega_k = b_u$) then
    begin
    if ($\nu < p_0 + p_1$) then
        $\omega_{k+1} = \omega_k$
    else
        $\omega_{k+1} = \omega_k - \delta$
    end.
if ($\omega_k$ is saturated to the lower bound, i.e., $\omega_k = b_l$) then
    begin
    if ($\nu < p_0 + p_2$) then
        $\omega_{k+1} = \omega_k$
    else
        $\omega_{k+1} = \omega_k + \delta$
    end.
```

Our workload generator is somewhat primitive and could be improved in terms of modeling or synthesizing real background workload. The exact reproduction of real background workload on real machines is not critical here as we only use the synthetic workload for evaluating our dynamic mapping strategy [2]. Note that our workload generator is a Markov process since the new background workload is based only on the background workload in the previous state. Further, this synthetic background workload can be reproduced because the seed to the random-number generator can be controlled.

The descriptors for all machines in our experiments are generated randomly. The lower and upper bounds on background workload in our experiments are set to 1 and 25. Note that if the background workload is 1, it means that the processor is totally dedicated to the ANN simulation, whereas if the background workload is 25, it means that only 4% of the processing time is for the ANN simulation. The detailed numerical values used for the descriptors in the experiments are presented elsewhere [2].

The experimental results are shown with respect to the 3-processor, 10-processor, 25-processor, and 100-processor multicomputers in Table 7.6. The gain g is defined as the gain in performance over the static one. Specifically, for N^{iter} iterations, the gain is defined as $g \triangleq T^{\text{dyn}}/T^{\text{static}}$, where T^{dyn} is the total simulated time in completing N^{iter} iterations of parallel ANN simulation plus the time expended in all the remappings in the dynamic case; T^{static} is the total simulated time in completing N^{iter} iterations of parallel ANN simulation plus one mapping time in the static case.

The gain efficiency $\hat{g}$ represents the goodness of the dynamic mapping strategy with respect to the best gain $g^{\max}$ obtained under the condition that remapping is carried out for each iteration and that the time expended in the mapping algorithm is negligible. Specifically, $\hat{g} \triangleq g/g^{\max}$.

The expected gains and gain efficiencies are shown in Table 7.6 with 95% *confidence intervals*. Each result was run for 10 iterations of the parallel ANN simulation. For the cases with 3, 10, and 25 processors, the number of samples is 100 each. For 100 processors, the number of samples is only 10 each due to its long simulation time.

We see in Table 7.6 that the gain is usually between 1 and 4, which is not significantly large. The reason is that the dynamic mapping strategy can gain significantly only when the background workload in each machine is changing rapidly all the time. We also see in Table 7.6 that the gain efficiencies are very high, indicating that our dynamic mapping strategy almost achieves full utilization of the resources in the multicomputer.

It is interesting to note that larger multicomputers generally have better gains but worse gain efficiencies. The reason for better gains is that a larger multicomputer is more likely to have an imbalance in workload; the reason for worse gain efficiencies is that the remapping overheads are relatively high.

TABLE 7.6
Summary of Simulation Results in Multicomputers with Dynamic Background Workload

		3 processors		10 processors		25 processors		100 processors	
ANN	Workload	$E[g]$	$E[\hat{g}]$	$E[g]$	$E[\hat{g}]$	$E[g]$	$E[\hat{g}]$	$E[g]$	$E[\hat{g}]$
FC-1	WL-1	1.46 ± 0.13	0.987 ± 0.001	1.93 ± 0.01	0.972 ± 0.001	2.43 ± 0.01	0.981 ± 0.001	3.43 ± 0.15	0.915 ± 0.018
FC-2	WL-2	2.07 ± 0.01	0.912 ± 0.002	2.24 ± 0.01	0.970 ± 0.001	2.69 ± 0.01	0.971 ± 0.001	4.12 ± 0.20	0.934 ± 0.017
FC-3	WL-3	1.10 ± 0.01	0.996 ± 0.001	2.36 ± 0.01	0.906 ± 0.001	2.83 ± 0.01	0.987 ± 0.001	4.33 ± 0.20	0.953 ± 0.013
ML-1	WL-4	1.42 ± 0.01	0.978 ± 0.001	1.66 ± 0.01	0.992 ± 0.001	2.37 ± 0.02	0.952 ± 0.002	3.92 ± 0.18	0.892 ± 0.019
ML-2	WL-5	1.22 ± 0.01	0.968 ± 0.001	2.09 ± 0.01	0.960 ± 0.001	1.98 ± 0.02	0.832 ± 0.003	4.08 ± 0.19	0.878 ± 0.019
ML-3	WL-6	2.24 ± 0.01	0.968 ± 0.001	2.28 ± 0.01	0.965 ± 0.001	2.25 ± 0.02	0.954 ± 0.001	4.35 ± 0.19	0.873 ± 0.015
NL-1	WL-7	4.17 ± 0.02	0.959 ± 0.001	2.09 ± 0.01	0.962 ± 0.001	2.12 ± 0.02	0.987 ± 0.001	3.46 ± 0.16	0.903 ± 0.019
NL-2	WL-8	2.01 ± 0.01	0.921 ± 0.001	1.79 ± 0.01	0.947 ± 0.001	2.08 ± 0.01	0.972 ± 0.001	3.70 ± 0.16	0.907 ± 0.021
NL-3	WL-9	1.59 ± 0.01	0.893 ± 0.002	2.20 ± 0.01	0.968 ± 0.001	2.22 ± 0.01	0.945 ± 0.001	3.76 ± 0.15	0.813 ± 0.018

Note. The results are shown with 95% confidence intervals of expected gains and gain efficiencies using our dynamic mapping strategy.

8. CONCLUSIONS

In this paper, we have studied the optimal mapping on a multicomputer for a multilayer artificial neural network based on a static learning algorithm. Processors in the multicomputer may be heterogeneous, have dynamic workload, and be connected by communication links of different speeds. The mapping problem is NP-hard in general and cannot be solved even for a network with a small number of neurons. We derive a number of results for simplifying the mapping problem so that neural networks with thousands of neurons can be mapped. Our results are useful for designing a special-purpose computer for ANN simulations and for determining the suitability of an existing computer system for ANN applications.

Our mapping algorithm is based on the observation that the computation time dominates the communication time in the learning process within a cluster of the neural network. By starting with a user-specified error tolerance, the mapping algorithm has three steps. First, the multicomputer is partitioned in such a way that the deviation in performance of a heuristic routing scheme from the optimal one can be bounded. Second, the neural clusters are mapped optimally on the partitions. Finally, heuristic routes are determined for neurons communicating within a partition. We show that the final error incurred by this algorithm is within the error tolerance specified by the user. Experimental results based on a 16-processor Intel iPSC/2 computer and a network of three Sun workstations are found to be very close to the analytical results predicted. Simulation results on systems with dynamic background workload show that our dynamic mapping strategy can almost always achieve full utilization of resources.

REFERENCES

1. Arlauskas, R. iPSC/2 system: A second generation hypercube. *Proc. 3rd Conference on Hypercube Concurrent Computers and Applications*, 1988, pp. 38–42.
2. Chu, L.-C. *Optimal Mapping of Neural Networks on Multicomputers*. M.Sc. thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, May 1991.
3. Close, P. The iPSC/2 node architecture. *Proc. 3rd Conference on Hypercube Concurrent Computers and Applications*, 1988, pp. 43–50.
4. Feldman, J. A., Fanty, M. A., Goddard, N. H., and Lynne, K. J. Computing with structured connectionist networks. *Comm. ACM* **31**, 2 (Feb. 1988), 170–187.
5. Forrest, B. M., Roweth, D., Stroud, N., Wallace, D. J., and Wilson, G. V. Implementing neural network models on parallel computers. *Comput. J.* **30** (1987), 413–419.
6. Garey, M. R., and Johnson, D. S. *Computers and Intractability*. Freeman, San Francisco, CA, 1979.
7. Ghosh, J., and Hwang, K. Mapping neural networks onto message-passing multicomputers. *J. Parallel Distrib. Comput.* **6** (1989), 291–230.
8. Graf, H. P., Jackel, L. D., and Hubbard, W. E. VLSI implementation of a neural network model. *Computer* **21,** 3 (Mar. 1988), 41–49.
9. Hwang, K., and Ghosh, J. Critical issues in mapping neural networks on message-passing multicomputers. *International Symposium on Computer Architecture*. ACM/IEEE, 1988, pp. 3–11.
10. Kung, S. S., and Hwang, J. N. Parallel architectures for artificial neural nets. *Proc. International Conference on Systolic Arrays*. IEEE, 1988, pp. 163–174.
11. Kung, S. Y., and Hwang, J. N. A unified systolic architecture for artificial neural networks. *J. Parallel Distrib. Comput.* **6** (1989), 358–387.
12. Lin, W.-M., Prasanna Kumar, V. K., and Wojtek Przytula, K. Algorithmic mapping of neural network models onto parallel SIMD machines. *Trans. Comput.* **C-40,** 12 (Dec. 1991).
13. Lippmann, R. P. An introduction to computing with neural nets. *Accoust. Speech Signal Process. Mag.* (Apr. 1987), 4–22.
14. McClelland, J. L., and Rumelhart, D. E. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, *Foundations*. Bradford Books, Cambridge, MA, 1985.

15. Nugent, S. The iPSC/2 direct-connect communication technology. *Proc. 3rd Conference on Hypercube Concurrent Computers and Applications*, 1988, pp. 51–60.

16. Pomerleau, D. A., Gsciora, G. S., Touretzky, D. S., and Kung, H. T. Neural network simulation at warp speed: How we get 17 million connections per second. *Proc. International Conference on Neural Networks*. IEEE, July 1988, Vol. II, pp. 143–150.

17. Ullman, J. D. NP-complete scheduling problems. *J. Comput. System Sci.* **10** (1975), 384–393.

18. Wah, B. W., and Chu, L. C. Efficient mapping of neural networks on multicomputers. *Proc. International Conference on Parallel Processing*. Pennsylvania State Univ. Press, Aug. 1990, Vol. I, pp. 234–241.

Received August 19, 1991; revised October 9, 1991; accepted October 18, 1991

LON-CHAN CHU is a Ph.D. degree candidate in the Department of Electrical and Computer Engineering at the University of Illinois, Urbana–Champaign, where he received the M.S. degree in 1991. His research interests include real-time scheduling for artificial intelligence, search, approximate processing, learning heuristics, parallel processing, and fault-tolerant neural networks.

BENJAMIN W. WAH is a professor of electrical and computer engineering at the University of Illinois, Urbana–Champaign. He has published extensively in the areas of computer architecture, parallel processing, artificial intelligence, and computer networks. He is a University Scholar of the University of Illinois and a Fellow of the IEEE.