Neural Networks for Fast Arbitration and Switching Noise Reduction in Large Crossbars

Joydeep Ghosh, Ajat Hukkoo, *Member, IEEE* and Anjun Varma, *Member, IEEE*

Abstract—A neural network-based controller is presented for the real-time arbitration of routing paths in large crossbar switches constructed from one-sided crosspoint chips. This controller is suitable for a synchronous environment where a number of connection requests are simultaneously presented to the switch. The controller aims to maximize the effective bandwidth of the switch and to minimize the simultaneous-switching noise in the individual chips. The controller uses multiple winner-take-all networks coupled with some competitive-cooperative mechanisms to achieve the joint optimization. The effects of various network parameters are studied through simulation, and cases leading to nonoptimal solutions analyzed. The results show that the arbitration complexity and time scale well with the size of the switches, and the throughput achieved is close to the theoretically maximum attainable. We also introduce a hierarchical neural network controller for a packet-switched environment where connections are established and broken asynchronously. This controller provides almost the same level of performance as the first one, but with significantly reduced computation for each connection request.

I. INTRODUCTION

CROSSBAR switches are used extensively as multi-processor interconnection networks and for communication switching. Although small crosspoint arrays are easy to implement on a single chip, several problems arise when the size is increased. Current VLSI and packaging technologies are capable of meeting the circuit-density and the pin-count requirements of large crosspoint arrays. For example, the single-chip implementation of a 72×72 -line crossbar has been reported [20], and an array of size 256×256 is considered feasible using multiple chips [3]. The primary limiting factor to the cost-effective single-chip realization of such large arrays now is the inductive noise generated by the line-drivers driving the output leads of the chip. When many of these drivers are active simultaneously, a substantial transient current passes through the inductance of the power distribution system, causing a noise spike to emerge on the power lines. This phe-

nomenon is known as the simultaneous switching noise or the *Delta-I* noise [4], [24]. The resulting fluctuation of the power-supply voltage level can cause false switching of devices with an accompanying loss of data.

While the *Delta-I* noise problem is present in all VLSI chips with a large number of output leads, it is particularly severe in large crossbar switching chips because they have a large number of data output lines. This problem can be alleviated by constructing crossbar networks using one-sided crosspoint switching chips [6], [28]. As explained in Section II, these networks allow a pair of ports to be connected using one of many available internal switching paths. By choosing the switching paths properly, it is possible to distribute the active off-chip drivers uniformly over the chip matrix, thus reducing the *Delta-I* noise in each chip to acceptable levels [29].

In a nonblocking interconnection network, any request to connect two idle ports can be satisfied without disturbing existing connections. For a one-sided crossbar with a limit on the allowable number of concurrently active line-drivers per chip, it is possible that a connection-request is rejected even when a switching path is available to make the desired connection. This reduces the effective overall network bandwidth. Situations in which this can happen are detailed in [8]. Thus smart path allocation policies are needed to maximize the number of requests that are fulfilled simultaneously for a given sequence of connection/disconnection requests. In summary, a joint optimization of network throughput and noise reduction is needed. Moreover, the time taken to reconfigure the network should be much smaller than the average duration of a connection request. This motivates the use of a highly parallel artificial neural network controller for path allocation and arbitration.

Hopfield-Amari networks and their variations have been widely used to solve combinatorial optimization problems in which the cost function can be expressed as a quadratic Lyapunov "energy function" [13], [14]. Several researchers have found, however, that Hopfield nets often perform poorly in practice with respect to computational requirements, convergence rate, and quality of solutions [27], [30]. The deterministic Hopfield model has been shown to be a mean-field approximation of a spin-glass, with the gain of the neural cells being proportional to the inverse of the temperature [22]. This suggests that better

Manuscript received July 9, 1990; revised November 30, 1990. This work was supported in part by an NSF Research Initiation Award MIP 9011-787. A. Hukkoo was supported by an MCD Fellowship. This paper was recommended by Associate Editor J. Vandewalle.

J. Ghosh is with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712-1084.

A. Hukkoo is with Intel Corp., Santa Clara, CA.

A. Varma is with the Computer Science Department, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

IEEE Log Number 9100876.

0098-4094/91/0800-0895\$01.00 ©1991 IEEE

solutions can be obtained by gradually increasing the gain until a step function is attained. Newer models that are better at avoiding local minima or have faster convergence rates have been proposed and compared [21], [23], and their architectural demands analyzed [7].

Recently, some researchers have studied applications of artificial neural networks to control problems in switching and communications [2], [12], [19]. The continuous Hopfield neural net model (also known as Grossberg additive model [10]) has been used to design a controller for a packet-switched two-sided crossbar switch [19]. The authors assume that packets are sent in batches. For each batch, the packet destinations are specified by a request matrix. The controller attempts to find a configuration matrix that has maximum possible overlap with the request matrix, so as to maximize the throughput. The entire crossbar is reconfigured after each set of packet transmissions. Under these assumptions, the authors found the average throughput efficiency to be over 98% for an 8×8 crossbar.

Brown used "winner-take-all networks" (WTA's) [5] as the basic control mechanism for two-sided crossbar switches, which are used as subcomponents in Clos networks and other multistage networks [2]. He introduced neural cell pairs with different time constants as a way of marking switches that have been set more recently. With this mechanism, the sequential algorithms for finding routing paths in multistage networks could be emulated in a macroscopically sequential manner by a neural network controller.

In this paper, we propose and evaluate neural network controllers for setting switches in large one-sided crossbar networks constructed from multiple chips. The controller problem here is much more involved than that for two-sided crossbars since there are multiple choices for the path selected for making a connection, and because of the need to spread the switching activity among all the chips. The paper is organized as follows: In Section II, we introduce one-sided crosspoint networks that provide added flexibility over conventional two-sided crossbar switches because of the presence of multiple paths between any source-destination pair. The parameters of the switching network are defined, and different ways of receiving and processing connection requests are outlined. Section III explains the controller architecture based on WTA's. In Section IV, experimental results for determining the effectiveness of the neural network controller are described, and cases that yield invalid solutions are analyzed. Section V presents an alternative controller that uses hierarchical neural net control mechanisms for processing asynchronous connection requests. Concluding remarks are given in Section VI.

II. ONE-SIDED CROSSPOINT NETWORKS

A conventional (two-sided) crossbar network consists of a set of input lines and a set of output lines, conceptually placed perpendicular to each other, with switches placed

Fig. 1. A two-sided crosspoint switch.

at each point where the lines cross. Large crossbars are implemented by partitioning the switching matrix into smaller rectangular blocks and assigning each block to a chip. Fig. 1 illustrates a two-sided crosspoint matrix with 16 inputs and outputs, constructed from 16 4×4 switching chips. Note that there is a unique path between a pair of input and output ports. This restriction can lead to an uneven distribution of the active drivers among the switching chips. For example, if the inputs to the first row of chips are connected to the output lines in the last column of chips, then all the drivers in the top-rightmost chip are used (see Fig. 1), but none of the drivers in the first row or last column of chips are active. Thus the chips in the network have to be designed to work reliably even in the event that *all* its line-drivers become active simultaneously.

An alternate way of designing crossbar networks is by means of one-sided crosspoint switching chips. A one-sided crosspoint matrix consists of a set of port-lines and a set of bus-lines placed perpendicular to each other, with switching elements placed at the points of intersection. A connection between two port-lines can be established through any of the internal (column) busses. In the full-duplex implementation, each port-line is actually two wires, one for communication in each direction. Similarly, each bus-line actually consists of two wires. The architecture and implementation details of one-sided crossbar networks are given in [6] and [8]. Fig. 2 shows a one-sided network with $N = 32$ input/output ports and 16 internal busses. The switch matrix is constructed out of 16 crosspoint chips, each of size 8×4 . Note that each horizontal line in Fig. 2 represents a full-duplex channel and each vertical bus a pair of bidirectional lines in a full-duplex implementation. A crucial feature of the one-sided network is the presence of multiple paths connecting any two ports. Any of the available column busses can be used to make the desired connection. This flexibility can be exploited by a smart bus arbitrator to distribute the active line-drivers more uniformly among the switching chips than is possible in a conventional two-sided design.

A connection between a source port and a destination port in the one-sided switch is established by locating an unused column-bus (internal bus) and then turning on the

Fig. 2. A one-sided crosspoint with 32 ports, with four connections distributed over different chips.

Fig. 3. Receivers and off-chip line drivers in a one-sided crosspoint switch.

two crosspoints where the source and destination rows intersect with the selected bus. If these two crosspoints are within the same chip, no column drivers need be activated. Otherwise the signal needs to be routed off-chip. This requires the activation of a column driver in the source-chip as well as the destination-chip to provide a full-duplex path. Fig. 3 shows the active drivers for the two cases. In case (a), a row-driver is activated for each active port and no column-drivers are activated. This is referred to as an *internal connection*. In case (b), a row-driver and a column-driver are activated for every active port. This is referred to as an *external connection*. Since any unused column bus can be selected for making a connection between two ports, the choice can be made to optimize various criteria. In particular, the connecting busses can be chosen so as to minimize the maximum number of drivers active in any chip at that instant.

In this paper, we consider one-sided crosspoint matrices with N ports and M internal busses, constructed from r rows and c columns of $n \times m$ chips. Thus $N = r \times n$ and $M = c \times m$. We use $chip(i, j)$, $0 \leq i \leq r - 1$; $0 \leq j \leq$

$c - 1$; to denote chip addresses, and $switch(i, j)$, $0 \leq i \leq N - 1$; $0 \leq j \leq M - 1$; to label the switch crosspoints. If the type is evident from the context, only the indices (i, j) are given. When all the ports are in use and all connections are external, the maximum number of off-chip column-drivers that are simultaneously active on any chip is at least $\lceil n/c \rceil$. This happens if these drivers are spread evenly. Let d be the *maximum allowable* number of active column-drivers per chip. This corresponds to an actual limit of $2d$ active drivers per chip, considering both row- and column-drivers. For nonblocking operation, the pertinent range of d is from $\lceil n/c \rceil$ to m . As d is made smaller, the difficulty of allocating paths in a given network increases, thereby reducing its throughput.

There are two modes of making connection requests to the controller. In a *synchronous* mode or *batch mode*, all crosspoints are deactivated initially. When a set of requests is made, the controller tries to simultaneously set up as many pairs of connections as possible within the given constraints. After the desired communication is over, a new set of requests is presented, and the entire network is reconfigured to cater to this new batch. Such batch requests are encountered in a multiprocessor system where the processors operate synchronously and submit their requests to the network simultaneously. An example of such a system is the IBM GF-11 supercomputer designed for efficient execution of quantum chromodynamics and related problems [1].

Alternatively, the switch can be used in *incremental mode*, wherein connection or disconnection requests from the ports are made asynchronously and independently. This is a more accurate model for many real situations such as requests for calls in a telecommunication system.

For the batch mode, the input to the controller is an $N \times N$ binary, symmetric *request matrix*, R . $R(i, j)$ is 1 iff a (bidirectional) connection is desired between ports i and j , and zero otherwise. For a given request matrix R , the maximum number of port pairs that can be connected is given by the cardinality of a maximal matching of the graph for which R is an adjacency matrix. Let this be $C_{\max}(R)$. Let the actual number of connections made by the controller be $C_{\text{batch}}(R)$. Then, the instantaneous throughput efficiency, η , is

$$\eta = \frac{C_{\text{batch}}(R)}{C_{\max}(R)} \times 100\%.$$

If we neglect the overheads in reconfiguring the switches, the average efficiency is obtained by finding the average of the instantaneous efficiency over the appropriate time period. This is one of the measures used to evaluate the quality of the assignments made by the controller.

The maximum load on the network occurs when all N ports are active. This is possible only if the connection requests have no conflicts. The corresponding request matrix has a maximum matching of size $N/2$. For a given

request matrix R , we define the offered network load as

$$\text{network load} = \frac{C_{\max}(R)}{N/2} \times 100\%.$$

In general, the difficulty of finding a bus-allocation satisfying the d -constraint increases with the offered network load. To impose maximum severity, the network load was set at 100% in our simulations unless mentioned otherwise.

The controller described in the next section maintains the number of active drivers in every chip within a given limit while allocating the internal busses to connection requests. This constrained allocation can reduce the throughput efficiency. However, as shown in the next section, such degradation is very small in the examples studied.

III. NEURAL NETWORK CONTROLLER ARCHITECTURE

3.1. Hopfield Nets and Winner-Take-All Mechanisms

Before detailing the controller architecture, we briefly describe two well-known neural network structures that will be used by the controller. The continuous Hopfield model [14] consists of an ensemble of neurons or cells, with cell i being connected to cell j through a "synapse" with transconductance strength $T_{i,j}$. The input activation, u_i , of cell i is governed by

$$C \frac{du_i}{dt} = \sum_{j \neq i} T_{i,j} V_j - \frac{V_i}{R} + I_i \quad (1)$$

where $V_j = g(u_j)$ is the output of cell j , and I_i is an external current (bias) to cell i . C and R are analogous to the capacitance and resistance, respectively, and determine the time constant for evolution of cell activity. If the $T_{i,j}$ matrix is symmetric and the activation function $g(x)$ is sigmoidal, then it can be shown that there is a Lyapunov energy or cost function that decreases monotonically to a (local) minima as the cells evolve according to (1). This has prompted the use of the continuous Hopfield network for solving combinatorial optimization problems [13], [21].

A WTA network [5], [9] is a connectionist mechanism that employs competitive learning [26] to identify the most active cell from among a group of N cells. Each cell competes with the others by sending positive feedback to itself and negative feedback (lateral inhibition) to all other cells, as shown in Fig. 4(a). If the thresholds and weights are chosen correctly, then eventually, only one cell—the winner—remains active while the outputs of all other cells decay to some quiescent value. The WTA circuit can be transformed into the one shown in Fig. 4(b), where the net output of all cells are collected and transmitted as a common inhibition signal to the cells. The resultant circuit needs only $O(N)$ instead of $O(N^2)$ interconnects, and can be efficiently fabricated using CMOS integrated circuits [16]. Moreover, the circuit can be read-

Fig. 4. A K -WTA circuit.

Fig. 5. Connectivity among different cells in a neural network arbitrator for one-sided switches with synchronous requests (batch mode).

ily extended to a " K WTA" network in which the K most active cells are selected [18]. We extend the schematic representation introduced by Brown [2] by denoting a K -WTA as in Fig. 4(c).

3.2. Controller for Batch Mode

The controller architecture shown in Fig. 5 closely follows the topology of the one-sided crosspoint switch. It uses one *switch cell* or "neuron" for each crosspoint, and additional cells for enforcing various constraints and optimization considerations. The switch cells are arranged in an $N \times M$ matrix, and labeled $(0,0)$ through $(N-1,$

$M-1$). This matrix is augmented by an extra column of *row-control cells*, labeled (i, M) , and a row of *column-control cells* (N, j) . Finally, there is a *chip cell* for each crosspoint chip that is used to monitor the net activity level on each chip. Thus there is a total of $(N+1)(M+1) + rc$ or $O(NM)$ cells. The control and chip cells are used to enforce certain constraints, while the switch cells directly determine whether the corresponding crosspoint is enabled or not.

The four types of cells have different activation functions and behavior. If the output of a switch cell is > 0.9 , then the corresponding crosspoint is enabled; otherwise, it is disabled. If a connection is made between ports i and j , using internal bus k , then the cells (i, k) and (j, k) should be the only switch cells activated in rows i and j , and in column k . The column cell for column k is used to enforce the second condition, while the first condition is enforced through the use of WTA's.

For each row, we implement a WTA network across the $M-1$ switch cells and the corresponding row cell. Row cell (i, M) is kept activated so long as port i wants to stay idle. Due to the competitive mechanism, this inhibits all the switch cells in row i and prevents them from getting activated. On the other hand, if port i is already connected or has an unfulfilled connection request, then row cell (i, M) is kept off, thus allowing some switch cell in row i to be a winner. The connectivity among the different cells is shown in Fig. 5.

The switch cells behave like continuous Hopfield-model cells [14]. The output of switch cell (i, j) is $V_{i,j} = g(u_{i,j})$, where $u_{i,j}$ is the input activation of that cell. $V_{i,j}$ takes on a value between 0 and 1, as determined by the following activation function:

$$\begin{aligned} g(u) &= 0, & \text{if } u \leq 0, \\ &= u, & \text{if } 0 \leq u \leq 1, \\ &= 1, & \text{if } 1 \leq u. \end{aligned} \quad (2)$$

This choice led to better results than commonly used sigmoidal activation functions such as the hyperbolic tangent or the logistic map: $f(x) = (1/1 + e^{-x})$.

For each switch cell column, we want either none or exactly two of the cells to be "on," and the rest to be "off." This is enforced by the column control cells, whose activation function is given by

$$\begin{aligned} V_{N,j} &= \sum_i V_{i,j}, & \text{if } \sum_i V_{i,j} \leq 1, \\ &= 2 - \sum_i V_{i,j}, & \text{if } \sum_i V_{i,j} \geq 1. \end{aligned} \quad (3)$$

The output of a column-control cell is used to inhibit all the switch cells in the same column, if the net activation of all switch cells in a column strays from zero or two.

A WTA network with $O(M)$ connections is implemented for every row of the cell matrix, using the row-control cell for that row. The output of the row-control cell is kept at zero if there is at least one 1 in the corresponding row of the request matrix. Otherwise, the

output of this cell is clamped to 1. Mathematically,

$$V_{i,M} = U_0 \left(\sum_j R(i, j) \right)$$

where U_0 is the unit step function. The WTA mechanism turns off all switch cells in row i if $R(i, j) = 0$, for all j , and allows one cell to be "on" otherwise.

Finally, the activation function of the chip cells is given by

$$g_{\text{chip}}(u_{a,b}) = U_0(u_{a,b} - d)$$

where

$$\begin{aligned} u_{a,b} &= \sum_{i=an}^{(a+1)n-1} \sum_{j=bm}^{(b+1)m-1} V_{i,j}, \\ a &= 0, \dots, r-1; \quad b = 0, \dots, c-1. \end{aligned}$$

In the above equations, d is the delta-1 noise limit for the chip, and U_0 is the unit step function.

The evolution of the switch cells is simulated in discrete time steps, using the update equations

$$u_{i,j}^+ = u_{i,j} + (\delta t)(-t_1 - t_2 - t_3 + t_4) \quad (4)$$

where

- i) $t_1 = A \times \sum_{j=0, j \neq i}^M V_{i,j}$. This implements the competitive mechanism among cells in the same row.
- ii) $t_2 = B \times g_{\text{chip}}(u_{i \bmod n, j \bmod m})$. This denotes the inhibition from the appropriate chip cell.
- iii) $t_3 = D \times V_{N,j}$. This is the penalty if the number of active cells in a column is neither zero nor two.
- iv) $t_4 = E \times \sum_k R(i, k) V_{k,j}$. If ports i and k are to be connected, i.e., $R(i, k) = 1$, then cells (i, j) and (k, j) support one another through a sigma-pi unit [5]. This tries to ensure that the connections made are in accordance with the request matrix, R . Note that (4) is a discretized version of the first-order differential equation governing the update of neurons in the continuous Hopfield model (1).

IV. EXPERIMENTAL RESULTS AND ANALYSIS

For our experiments, the time increment δt was 10^{-2} . A smaller increment results in longer convergence time without significant improvement in the results. The choice of the coefficients, (in our case A , B , D , and E) is quite critical to the performance of the network, as has been observed by several researchers [11], [25]. After some fine-tuning, the values of A , B , D , and E chosen were 25, 10, 15, and 30, respectively. Request matrices of several sizes were generated through Monte-Carlo techniques. For each size, 1000 matrices were generated, and the results averaged over these instances. The simulations were performed on a Sun-4, and the user times given in Table I are the net times taken (including request generation and set-up times) averaged over the entire run of 1000 instances.

Initially, each switch cell has a low, randomly chosen activation value, with a mean of $1/N$. The controller

TABLE I
RESULTS OF BATCH MODE SIMULATIONS AT ABOUT 100% NETWORK LOAD

Crossbar Size		Chip Size	d limit	No. of iter. (median)	Convergence (%)	Throughput efficiency, η (%)	User time (ms)
ports, N	int. busses, M						
16	8	4×4	2	11	100	100	81.1
16	12	4×4	2	14	98.8	99.7	329.9
32	16	4×4	2	13	100	100	897.9
32	24	4×4	2	15	100	100	1853.8
32	24	8×4	2	16	97.7	99.7	2521.5
64	32	16×8	4	19	78.1	98.63	16592

network is simulated through a sequence of iteration cycles. In each iteration, every switch cell is selected exactly once, and its output value is updated according to (4). The cell to be updated next is chosen in a random but equitable manner as follows: A signature register of size $p = \lceil \log_2(N \times M) \rceil$ is used to implement a primitive polynomial of $GF(2^p)$ [17]. The polynomial generates all numbers from 1 to $2^p - 1$ in some order depending on the seed chosen, and these numbers determine the cell to be updated next. At the end of each iteration, the chip cells are examined and updated to reflect the change in change in activity of the switch cells.

The switch cell updates are stopped when each requesting row has a cell with output 0.9 or more. If this does not happen in 100 iterations, we presume that the network fails to converge, and start again with a new set of random values for the cell activations.

Table I presents the simulation results. To calculate the throughput efficiency, a maximal matching algorithm is run for each request matrix to obtain $C_{\max}$ which is then compared with C_{batch} obtained from the connection matrix. We notice that the efficiency is almost 100%, showing that the quality of the solutions is good. The request matrices were such that the network load was 100% in almost all cases. Since the d limit severely restricts the choice of internal busses once many ports are active, getting a high throughput is not trivial.

Even though the convergence rate is rather high as compared with many other applications using Hopfield-type networks, we did encounter cases when the network failed to converge. The frequency of this happening increased when there were more column busses available to make a connection, so that the search space was not constrained enough. By observing the evolution of cell activations for these cases, two cases were discerned.

- 1) The controller would oscillate between choosing columns k and l for implementing a connection between ports i and j . This could happen when $v_{i,k} \approx v_{j,l} \approx 1$; $v_{i,l} \approx v_{j,k} = a < 1$, and all other cells in rows i, j and columns k, l were off. This situation is expressed in Fig. 6(a). Such situations could be overcome by increasing the positive coupling along the columns (term t_4 in (4)) and increasing the inhibition of cells in the same row (term t_1).
- 2) There were a few situations where more than two cells were quite active in one column whereas another column was not utilized at all, as depicted in

Fig. 6. (a) Nonconvergence for batch mode; Case 1. (b) Nonconvergence for batch mode; Case 2.

Fig. 6(b). Such situations could be alleviated by increasing the constant A in term t_1 .

Table I also shows the median number of iterations required for convergence against the network size, given by the number of crosspoints. In a serial simulation, the time taken per iteration is $O(NM)$. The number of iterations required for convergence is experimentally observed to increase sublinearly with the network size. Thus the computational requirements scale well with increase in problem size. In a parallel implementation with one processor per cell, the time taken per iteration is almost independent of the network size. This implies that a VLSI implementation of the controller has the potential of performing bus arbitration in time quite impervious to the crossbar size.

V. A HIERARCHICAL CONTROLLER

In the incremental mode, a pair of entries in the R matrix is modified whenever a new request is received, without affecting the other entries. For this mode, a smaller and more efficient neural network can be used in

Fig. 7. A hierarchical controller for the incremental mode.

a two-level controller architecture. The key observation is that, within a given chip column, all the available internal busses are equivalent in their effect on Delta-1 noise or on system throughput. Similarly, these parameters are independent of the specific port among the set of ports accessing a particular row of chips, that is, making a connection request. Therefore, one can reduce the problem of selecting an internal bus to selecting the appropriate pair of chips (in the same chip column) used to make a connection. Once this is done, any of the available internal busses running through these two chips can be selected.

The proposed architecture thus has two levels, as shown in Fig. 7. The top level uses a neural network that has rc chip block cells arranged in an $r \times c$ matrix augmented by an extra row and column of control cells. If we need to connect or disconnect port i' to j' , then an incremental $r \times c$ request matrix, R' , is created, where $R'(i, j) = 1$ iff $i = \lfloor i'/n \rfloor$ and $j = \lfloor j'/m \rfloor$, and is zero elsewhere. Note that $R'(i, i) = 1$ is possible, and corresponds to the case where an internal connection is made.

Each block cell has a memory counter $M_{i,j}$, which stores the current number of active drivers on the corresponding chip. The update equation used for a block cell is given by

$$u_{i,j}^+ = u_{i,j} + (\delta t) \left(-A' \sum_{j=0, j \neq i}^c V_{i,j} - B_1 M_{i,j} - B_2 U_0(M_{i,j} - d) - D' V_{r,j} + E' R'(i, k) V_{k,j} \right). \quad (5)$$

Note that each cell receives an inhibition $B_1 M_{i,j}$ of magnitude proportional to the activity of the corresponding chip. It also receives a strong inhibition, given by $B_2 U_0(M_{i,j} - d)$ if the chip has already reached the d -limit. The row and column control cells behave as before,

except that now they get inputs from block cells instead of switch cells.

For the second (lower level) controller, a digital mechanism, outlined in [8] is used. This controller receives a signal from the neural net arbiter once the chips are chosen, and then activates the appropriate crosspoint pair after selecting any one of the available busses. Similarly, when a release request arrives, this controller deactivates a crosspoint pair, and decrements the corresponding two $M(i, j)$ values by one. The exception is for an internal connection involving row i , in which case $M(i, i)$ is decremented by 2.

For our experiments, the time increment δt was 0.05, and typical values for the coefficients A' , B_1 , B_2 , D' , and E' were 50, 25, 100, 50, and 75, respectively. Connection requests were generated using Monte-Carlo techniques until the desired capacity was reached. After that, a randomly chosen connection was removed and another connection chosen from among the free ports. For the special case of 100% network load, i.e., when all ports were busy, new connection/disconnection requests were generated by performing pairwise interchange between two pairs of communicating ports.

Table II summarizes the performance of the top level arbiter. The individual crosspoint chips are taken to be of size 64×32 , and the d limit was chosen as $\lfloor n/c \rfloor + 1$. Thus, even though only one more off-chip column driver was allowed than the lower-bound, 100% convergence was observed for almost all cases. The average convergence times are also very low. We observe that the time taken to converge is quite independent of the load or capacity at which the network is operated. The only cases in which the neural network did not converge to a valid solution occurred when the inhibitory terms were strong enough to cause all the cells to become inactive. This situation could be alleviated in two ways:

- 1) by adding an extra inhibitory term, $C'(\sum_{i,j} V_{i,j} - 2)$, which tends to enforce that exactly two cells be active. A value of $C' = 50$ was found to be suitable; or
- 2) by scaling B_1 in inverse proportion to the d limit.

To observe the performance of the controller in a crosspoint matrix with no extra chip columns, we simulated incremental operation of networks with such configurations. In all the examples studied, the chip-size was fixed at 48×24 and the number of chip-columns was chosen identical to the number of rows. This forces all the busses to be used when the network load is 100%. The results are shown in Table III for various network sizes. Here, the d limit is set to $\lfloor n/c \rfloor$, the best achievable, in the first table entry for each switch size. First, a series of connection requests are generated till the desired loading level is achieved. Then, a connection is picked at random for removal, followed by a request for a new connection among two of the free ports that belong to different chip rows. The results are recorded after 10000 connections/disconnections.

TABLE II
RESULTS OF THE INCREMENTAL MODE SIMULATIONS USING A HIERARCHICAL CONTROLLER

Crossbar Size		d Limit	Network Load (%)	Conver- gence (%)	User Time (ms)
Ports, N	Chip Cols, c				
256	5	14	70	100	4.18
256	5	14	80	100	4.62
256	5	14	90	100	4.62
256	5	14	100	100	3.70
384	7	10	70	100	13.75
384	7	10	80	100	14.75
384	7	10	90	100	13.75
384	7	10	100	100	11.50
512	9	9	70	100	35.56
512	9	9	80	100	38.63
512	9	9	90	100	42.75
512	9	9	100	100	36.91
768	13	5	70	100	166.13
768	13	5	80	100	171.19
768	13	5	90	100	163.94
768	13	5	100	100	168.19
1024	17	4	70	100	443.94
1024	17	4	80	100	435.24
1024	17	4	90	99	444.18
1024	17	4	100	95	463.76

TABLE III
INCREMENTAL MODE SIMULATION FOR NETWORKS WITH THE MINIMUM NUMBER OF CHIP COLUMNS

No. of Ports	No. of Chip Columns, c	d	Convergence			$d_{\max}$		
			Network Load					
			80%	90%	100%	80%	90%	100%
480	10	5	100	100	97	5	5	6
480	10	6	100	100	100	6	6	6
480	10	7	100	100	100	7	7	7
576	12	4	100	100	83	4	5	6
576	12	5	100	100	100	5	5	5
576	12	6	100	100	100	6	6	6
672	16	3	99	93	95	4	5	4
672	16	4	100	99	99	4	5	4
672	16	5	100	100	100	5	5	5

TABLE IV
COMPARISON OF NEURAL NETWORK AND BEST-FIT CONTROLLERS FOR REDUCTION OF SWITCHING NOISE

No. of Ports	No. of Chip Columns, c	$d_{\max}$ Neural Controller			$d_{\max}$ Best-Fit Controller		
		Network Load					
		80%	90%	100%	80%	90%	100%
480	10	5	5	6	5	6	10
576	12	4	5	6	5	6	10
672	16	4	5	4	4	6	8

Since the parameter d provides only a soft constraint in (4), it is possible that the number of active column-drivers actually reached in a chip exceeds d . The maximum of the observed number of active drivers/chip during each simulation is given as $d_{\max}$ in Table III. For large networks, these values closely matched the best possible.

Among the traditional controllers, such as those using the first-fit and best-fit algorithms [8], the best performance is observed to be that of a best-fit algorithm that

selects a chip column j for connecting a port is chip row i to one in row k , according to

$$\min(\max(M_{i,j}, M_{k,j})).$$

This algorithm attempts to keep the highest number of active drivers in a chip as low as possible by avoiding the use of chips with maximum number of active drivers for a new connection unless there is no other choice. We compared the performance of the neural network con-

Fig. 8. Average reconfiguration time versus network size for the incremental mode.

troller with that of the best-fit algorithm described above. The results are shown in Table IV. For each switch size and loading value, we recorded the maximum d reached over a sequence of 10000 connections/disconnections for both controllers. We observe that the controllers are comparable at lower loads, but the neural network controller is distinctly superior when the network loading approaches 100%.

In a parallel implementation of the neural network controllers, with one processor per cell, the convergence time is indicated by the number of iterations in the corresponding serial simulation. Fig. 8 shows estimated average time taken to converge in the incremental mode. We observe experimentally that the arbitration time scales almost linearly with network size. This makes a VLSI implementation attractive. The 8×8 neural network arbitrator described in [19] uses a Hopfield model and $2\text{-}\mu\text{m}$ CMOS technology, and typically computes configurations within 120 ns. We expect the hierarchical controller presented in this paper to fulfil a single connection request in $O(10^2)$ ns for networks with $O(10^2)$ ports. More specific timing estimates demand an actual implementation of the controller in VLSI and are dependent on technological parameters.

VI. CONCLUDING REMARKS

In this paper, we presented two neural network architectures for the controller of multiple chip crossbar networks, and analyzed their performance for batch and incremental connection requests, respectively. Since each communication line in a one-sided network is established through the activation of a *pair* of crosspoints, and since there are several choice of these pairs (one per available internal bus), the controller designs are much more involved than the one given in [19].

The hierarchical controller operates faster and provides better results than the batch controller. This is primarily because the reduced size of the neural network makes the number of constraints that need to be simultaneously

satisfied smaller. For combinatorial optimization problems such as the traveling salesman problem, where the multiple symmetries in the problem lead to the formation of several well-spaced global minima, poor convergence rates for the Hopfield net have been observed [30]. The application considered in this paper does not suffer from this disadvantage, and converges to valid solutions in most trials. Moreover, the quality of the solutions is typically superior to those obtained by more conventional controllers using first-fit or best-fit schemes [8].

When the simultaneous switching noise is the limiting constraint on the size of a crosspoint chip, an effective controller allows the synthesis of larger crossbar networks than would be possible otherwise. Moreover, the massive parallelism afforded by neural networks can be exploited to meet the fast reconfiguration demands of these switches. The WTA network, a key component of the arbiter architecture, has already been fabricated on CMOS integrated circuits using $O(N)$ of interconnects [16]. A VLSI implementation of the controller facilitates the design of a large crossbar switch serving as an interconnection network for highly parallel multiprocessor systems.

ACKNOWLEDGMENT

The authors thank the anonymous referees for their comments. They also thank Prof. J. Abraham for his suggestions on Hukkoo's thesis [15].

REFERENCES

- [1] J. Beitem, M. Denneau, and D. Weingarten, "The GF11 super-computer," in *Proc. Ann. Symp. Computer Architecture*, pp. 108-115, 1985.
- [2] T. X. Brown, "Neural networks for switching," *IEEE Communications Mag.*, pp. 72-81, Nov. 1989.
- [3] B. F. Colbry *et al.*, "High performance crosspoint switch implemented in CMOS on AVP," in *ICCD*, pp. 113-119, 1987.
- [4] G. Ditlow and A. Brown, "The Delta-1 simultaneous switching problem," Tech. Rep. RC 10015, Computer Sci. Dept., IBM T. J. Watson Research Center, Yorktown Heights, NY, May 1985.
- [5] J. A. Feldman and D. H. Ballard, "Connectionist models and their properties," *Cognitive Science*, vol. 6, pp. 205-254, 1982.
- [6] C. J. Georgiou, Fault-Tolerant Array of Cross-Point Switching Matrices, U.S. Patent 4 605 928.
- [7] J. Ghosh and K. Hwang, "Mapping neural networks onto message-passing multicomputers," *J. Parallel and Distributed Computing*, vol. 6, pp. 291-330, Apr. 1989.
- [8] J. Ghosh and A. Varma, "Reduction of simultaneous switching noise in large nonblocking crossbars," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 86-99, Jan. 1991.
- [9] S. Grossberg, "Neural expectation: Cerebellar and retinal analogues of cells fired by unlearnable and learnable pattern classes," *Kybernetik*, vol. 10, pp. 49-56, 1972.
- [10] —, "Non-linear neural networks: Principles, mechanisms and architectures," *Neural Networks*, vol. 1, pp. 17-61, 1988.
- [11] S. U. Hegde, J. L. Sweet, and W. B. Levy, "Determination of parameters in a Hopfield/Tank computational network," in *Proc. IEEE Conf. Neural Networks*, vol. 2, pp. 291-298, 1988.
- [12] A. Hiramatsu, "ATM communications network control by neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 122-130, Mar. 1990.
- [13] J. J. Hopfield and David W. Tank, "Neural computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141-152, 1985.
- [14] John J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," in *Proc.*

- National Academy of Sciences, USA*, vol. 81, pp. 3088–3092, May 1984.
- [15] A. Hukkoo, "Neural network controllers for bus-arbitration in crossbar switches," Master's thesis, Dept. Elect. Comp. Engr., Univ. of Texas, May 1990.
 - [16] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of $O(N)$ complexity," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 703–711.
 - [17] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
 - [18] E. Majani, R. Erlanson, and Y. Abu-Mostafa, "On the k-winners-take-all network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 634–642.
 - [19] A. Marrakchi and T. Troudet, "A neural net arbiter for large crossbar packet-switches," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1039–1041, 1989.
 - [20] R. F. Miracky *et al.*, "Design of a 64-processor by 128-memory crossbar switching network," in *Proc. ICCD*, pp. 526–532, Oct. 1988.
 - [21] C. Peterson, "Parallel distributed approaches to combinatorial optimization: Benchmark studies on traveling salesman problem," *Neural Computation*, vol. 2, pp. 261–269, 1990.
 - [22] C. Peterson and J. R. Anderson, "Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem," *Complex Systems*, vol. 2, pp. 59–89, 1988.
 - [23] C. Peterson and B. Soderberg, "A new method for mapping optimization problems onto neural networks," *Int. J. Neural Systems*, vol. 1, pp. 3–22, 1989.
 - [24] A. J. Rainal, "Computing inductive noise of chip packages," *AT&T Bell Laboratories Tech. J.*, vol. 63, pp. 177–192, Jan. 1984.
 - [25] J. Ramanujam and P. Sadayappan, "Optimization by neural networks," in *Proc. IEEE Conf. Neural Networks*, vol. 2, pp. 325–332, 1988.
 - [26] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Science*, vol. 9, pp. 75–112, 1985.
 - [27] B. Shirazi and S. Yih, "Critical analysis of applying Hopfield neural net model to optimization problems," in *Proc. IEEE Conf. Systems, Man, Cybernetics*, 1989.
 - [28] A. Varma, J. Ghosh, and C. Georgiou, "Reliable design of large crosspoint switching networks," in *Proc. 18th Int. Symp. Fault-Tolerant Computing*, pp. 320–327, June 1988.
 - [29] —, "Rearrangeable operation of large crosspoint switches," *IEEE Trans. Commun.*, vol. 38, pp. 1616–1624, Sept. 1990.
 - [30] G. V. Wilson and G. S. Pawley, "On the stability of the traveling salesman problem algorithm of Hopfield and Tank," *Biol. Cybern.*, vol. 58, pp. 63–70, 1988.

Joydeep Ghosh, for a biography, please see page 99 of the January 1991 issue of this TRANSACTIONS.

Ajat Hukkoo received the B.T. degree in electrical engineering from the Indian Institute of Technology, Bombay, India, in 1988, and the M.S. degree in electrical and computer engineering from the University of Texas, Austin, TX, in 1990.

He is currently with the Design Technology Department at Intel Corp., Santa Clara, CA, and is working on device level synthesis tools. His primary areas of interest include design automation tools for VLSI design. He is also interested in research topics such as neural networks and parallel processing applied for VLSI design.

Anujan Varma (S'83–M'85–S'85–M'86), for a biography, please see page 99 of the January 1991 issue of this TRANSACTIONS.