On-Line Prediction of Multiprocessor Memory Access Patterns

M.F. Sakr[1,2], C. L. Giles[1,5], S. P. Levitan[2], B. G. Horne[1], M. Maggini[4], D. M. Chiarulli[3]

[1]NEC Research Institute, Princeton, NJ, 08540
sakr@research.nj.nec.com
[2]EE Department, University of Pittsburgh, Pittsburgh, PA, 15261
[3]CS Department, University of Pittsburgh, Pittsburgh, PA, 15260
[4]Universita di Firenze Dipartimento di Sistemi e Informatica Via di Santa Marta, 3, 50139 Firenze, Italy
[5]UMIACS, University of Maryland, College Park, MD 20742

ABSTRACT

A neural network based technique is introduced which hides the control latency of reconfigurable interconnection networks (INs) in shared memory multiprocessors. Such INs require complex control mechanisms to reconfigure the IN on demand, in order to satisfy processor-memory accesses. Hiding the control latency seen by each access improves multiprocessor performance significantly. The new technique hides control latency by employing a time-delay neural network (TDNN) as a prediction technique that learns the current processor-memory access patterns and predicts the need to reconfigure the IN. Training and prediction of the TDNN is performed on-line. Based on three experiments, the TDNN is able to learn repetitive patterns and predict the need to reconfigure the IN thus, effectively hiding control latency of processor-memory accesses.

1 Introduction

Large scale multiprocessor systems need low-cost, highly-scalable, and dynamically reconfigurable interconnection networks (INs) [8]. Such INs offer a limited number of communication channels which are configured on demand to satisfy required processor-memory accesses. An IN controller is required to determine the IN configuration based on processor requests. Hence, the end-to-end latency incurred by such INs can be characterized by three components: *control time*, which is the time needed to determine the new IN configuration and to physically establish the paths in the IN; *launch time*, the time to transmit the data into the IN; and *fly time*, the time needed for the message to travel through the IN to its final destination. Launch time can be reduced by using high bandwidth opto-electronic INs, fly time is relatively insignificant in such an environment since the end-to-end distances are short. Therefore, control time dominates the communication latency.

In a demand driven environment, a processor accessing a memory module makes a request to the IN controller to establish a path (reconfigure the IN) that satisfies the processor's request. In a multiprocessor system executing a parallel application, the memory-access requests made by the processors follow a pattern based on the application. The goal of this work is to provide a technique that predicts a processor's request and performs the IN configuration prior to that request, thus hiding the control latency. To accomplish this, the predictive technique must learn the processor-memory access patterns and predict changes in the pattern. The effect of hiding control latency is to reduce the major component of communication latency in multiprocessor systems.

In this paper we examine how neural networks perform at predicting processor-memory access patterns to aid in reconfiguring an IN in a shared memory multiprocessor environment. While other prediction mechanisms could be used, neural networks have been quite successful as nonlinear predictors [9]. Our experiments use simulated on-line neural network learning and prediction using the memory access patterns of three parallel applications: *temperature propagation*, *matrix multiply*, and *1-D FFT*. The next section presents the environment of our experiment where we describe the shared memory multiprocessor model and the use of neural networks as predictors. In section 3, we present the organization of our experiments. Finally, we discuss our results and make projections about future directions of research.

2 Model

The shared memory multiprocessor (SMM) model used consists of **N** processors, **D** memory modules, a reconfigurable IN and an IN controller (Figure 1). Such INs can be configured to achieve any path between a processor and a memory module. However, at any given time only a subset of these paths are available. Because of contention for paths, the IN must be dynamically reconfigured to satisfy the set of current processor-memory accesses. In a typical

demand driven environment, when a processor needs to access a memory module it issues a request to the IN controller. The controller receives requests from all processors and reconfigures the IN to provide the paths necessary to service these requests.

The SMM model used in this paper employs an IN control system based on a different paradigm, that of *state sequence routing* (SSR) [2]. This paradigm takes advantage of the locality characteristics exhibited in memory access patterns [3] and reconfigures the network through a fixed set of configurations in a repetitive manner. The set of IN configurations consists of sets of compatible (non-blocking) paths called a state sequence. The IN controller, used for state sequence routing, consists of a *state generator* which is controlled by a *state transformer* (Figure 1). The state transformer, based on processor requests, determines the set of configurations. The state generator broadcasts this fixed length state sequence repetitively to each of the processors, memory modules, and switching elements of the IN. Thus, a processor that needs to access memory issues a *fault* (or a request) to the state transformer only if the current state sequence does not already include the required path to a memory module. In response to the fault, the state transformer adds the required path to the state sequence, possibly by removing an existing path.

SSR based control differs from demand driven control in that the IN controller needs only to respond to the changes in the memory access pattern and establish the initial paths; it is not required to respond to individual memory access requests. The state sequence router exploits the memory access locality inherent in these patterns by re-using the sequence of states, or paths, repetitively. Using SSR the average control latency, L, incurred by each access can be shown to be: $L = [(1-p)(k/2)]+[p(f+k/2)]$ where p is the probability of a fault, k is the sequence length, and f is the fault service time [2]. If the required path exists in the state sequence, there is no fault and the latency is just the time for the path to come around in the sequence. However if the path does not exist, a fault must be generated and serviced before the memory access can occur.

Our goal is to provide a technique that reduces the probability of a fault by predicting changes in memory access patterns and informing the controller of a needed transformation before a fault occurs. Thus, the controller is able to transform the state sequence to include the soon-to-be-needed paths avoiding the latency incurred by the fault.

Since the processor-memory access patterns change dynamically and thus can be modeled as a time series, any prediction mechanism that learns the pattern can eventually predict changes in the pattern. However, any mechanism would need to maintain a history of the memory access behavior. Therefore for this preliminary investigation, we chose to use a simple time delay neural network (TDNN) [4].

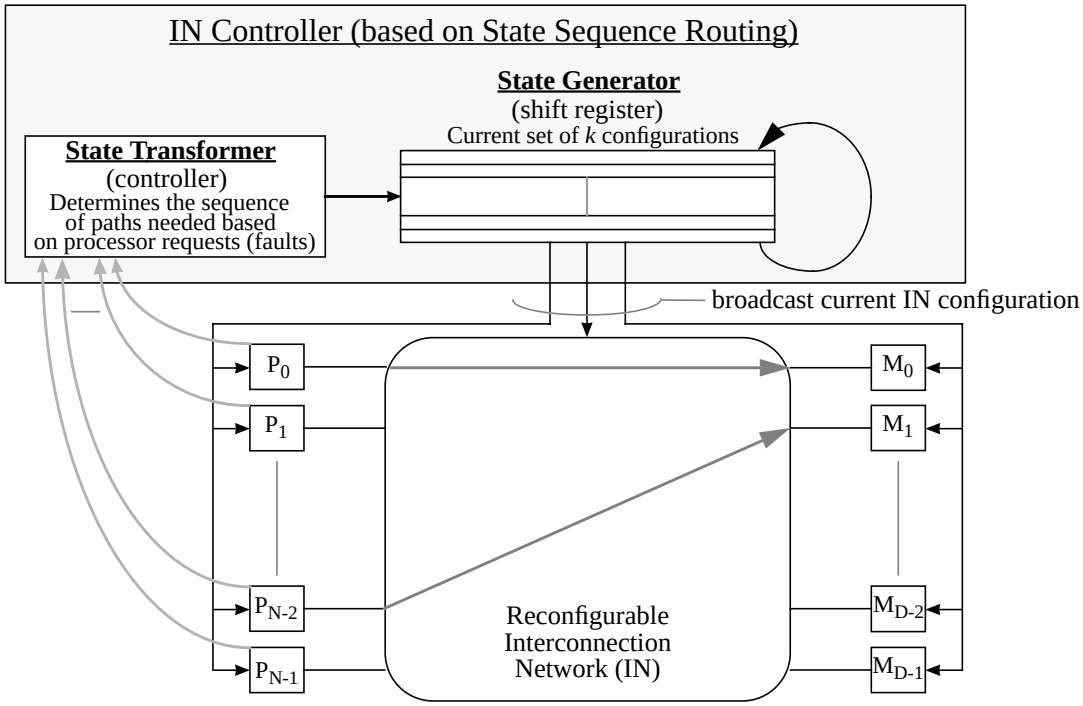

Fig. 1: A shared memory multiprocessor system.

3 Experimental Procedures

We performed three experiments to evaluate our technique [7]. Since our simulation environment is trace driven, each experiment consists of three distinct phases: First, we generate the raw memory traces of a parallel program and translate these traces into memory access patterns. Second, we use the memory access patterns as input to a TDNN to perform on-line training and prediction. Third, we evaluate the neural network predictions by simulating the multiprocessor behavior with and without the TDNN predictions. A more detailed description of each phase can be found in [6].

3.1 Extraction of Access Patterns

Using a trace driven shared memory multiprocessor (SMM) simulator [1], we generate the raw processor-memory accesses of a parallel program run on an 8x8 multiprocessor. Since the SMM simulator only provides relative memory access times for each processor, the accesses are serialized and then translated to a binary matrix form using a windowing mechanism illustrated in Figure 2. The top time line of Figure 2 depicts the serialized references. All references taking place during a fixed time window are combined in a single processor-memory access matrix. For example, if processor P5 accessed memory M3 at least once in the time period covered by a window then a "1" is placed in the matrix at column 5, row 3. For these experiments, large window sizes were used to compress the information in the traces.

This mechanism increases the relative number of changes in the pattern which was found to be necessary for the TDNN to learn these memory access patterns. Also, the on-line learning and prediction task is simplified by using an independent TDNN for each processor. Therefore, each TDNN trains on and attempts to predict sequences of memory access vectors of a single processor.

3.2 Neural Network Training and Prediction

The same TDNN architecture is used for all three experiments. The number of input neurons is 8 since each processor accesses 8 memory modules. Preliminary experimentation showed that a tapped delay line of length 5 suffices to give good prediction performance. Therefore the total number of input neurons is 48 ($8 \times (1 \text{ input} + 5 \text{ taps})$). Preliminary testing showed that a single hidden layer of 10 neurons is large enough for good prediction performance. The number of output neurons is 8 since the TDNN predicts the access of a single processor to 8 memory modules.

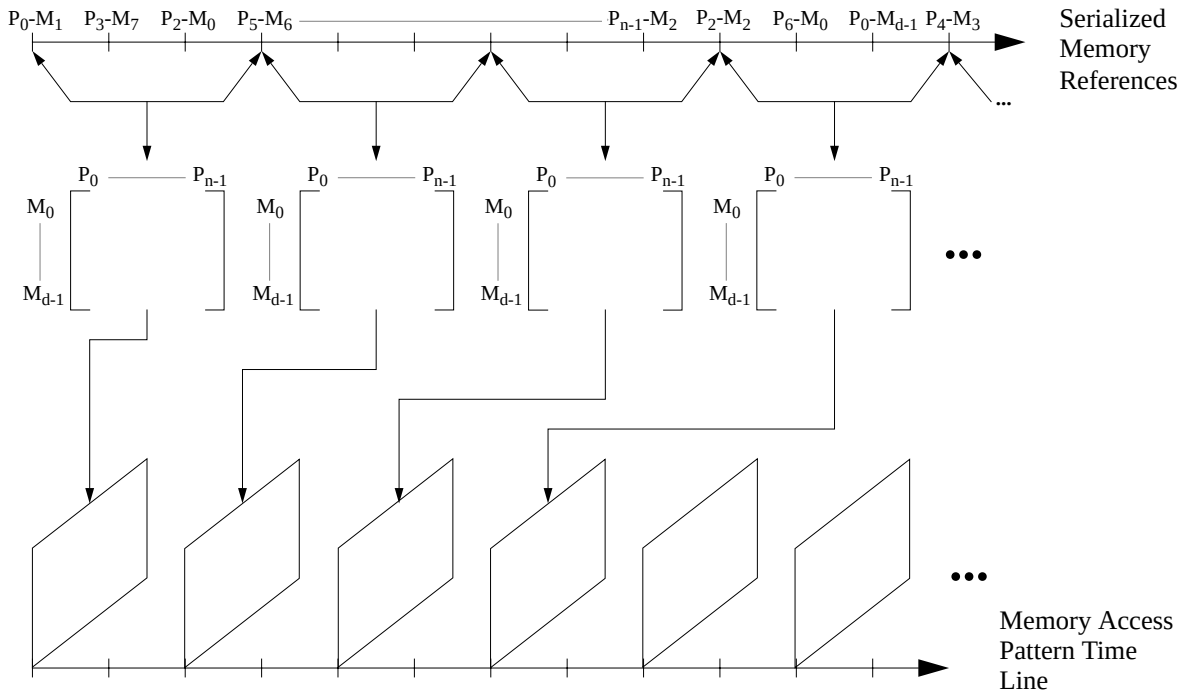

Fig. 2: Translation of the serialized memory traces into a matrix memory access pattern.

A TDNN on-line simulator [5] trains at each time step using the following parameters: *weights* are initialized using a uniform random distribution in the interval [-1/fan_in, 1/fan_in], where fan_in is the number of connections that enter a neuron; *bias weights* are used in the hidden and output layers only; total number of weights is 2282; a *hyperbolic tangent activation function* is used for hidden neurons and *linear activation function* for output neurons; *training set size* is a memory access vector that is trained on every time step; The training set is covered only once (*max # of epochs* = 1); *learning rate* is set to 0.01.

Learning is delayed by one time step since the target output for each input is available after one time step (one step prediction). All simulations discussed below use the same parameters. Having the same neural net architecture for all problems facilitates future hardware implementation.

3.3 Prediction Evaluation

The TDNN predictions are used as hints to the SSR while routing the memory access data. The number of IN faults incurred while performing the routing is logged and compared to the number of faults incurred without using the TDNN predictions.

4 Experimental Results

4.1 Temperature Propagation(2D Relaxation Algorithm)

The first program is a temperature propagation/ 2D relaxation algorithm. This parallel program is written for the 8×8 SMM simulator discussed earlier. The TDNN simulator is trained on the memory access pattern generated by a time window of length 5000, which is large enough to emphasize the changes in the pattern (Figure 3a).

The SSR performs the routing of the actual memory accesses while using the neural network predictions as hints to the SSR. Figure 3b depicts the (time averaged) number of faults incurred while routing the memory accesses with and without the TDNN predictions. This plot shows a ramp like shape for the initial 100 time steps since the time averaging window used here is of length 100.

For this program, the memory access pattern is stair-like (Figure 3a), where each discontinuity depicts a change in the memory access pattern. For the TDNN to be effective, it has to predict the “steps” in the access pattern because during the “stairs” the state sequence satisfies all accesses. As Figure 3b illustrates, the TDNN predictions greatly reduce the number of faults incurred. Hence, the TDNN is capable of predicting the changes in the memory access patterns thus removing the latency of servicing these faults.

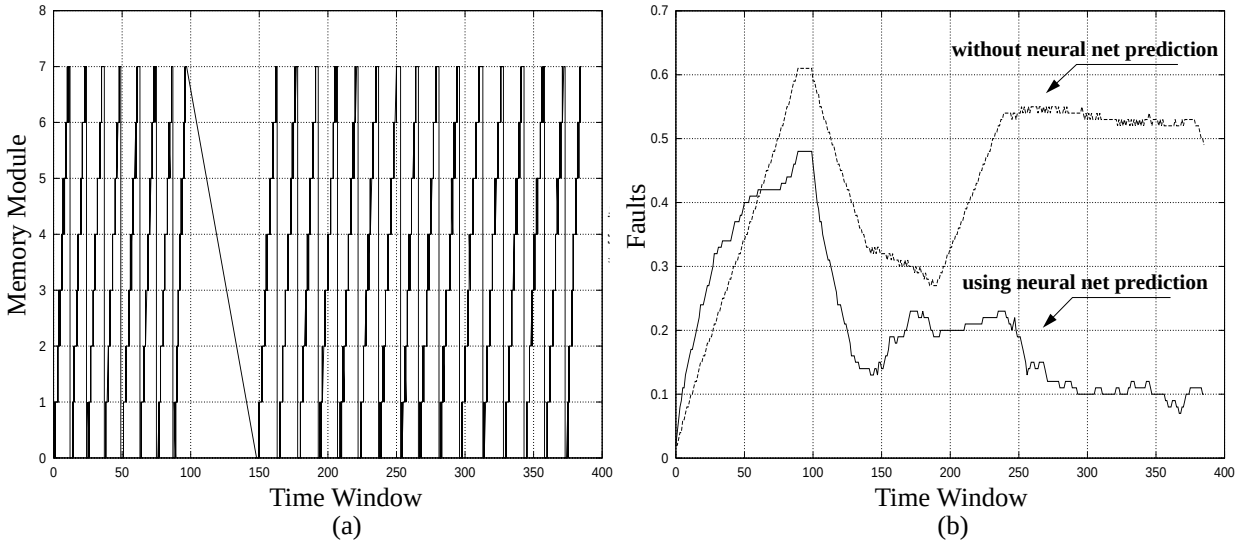

Fig. 3: (a) The memory access pattern of the temperature propagation program. (b) The number of network faults incurred with and without the TDNN predictions (time averaged).

4.2 Matrix Multiply

Our second application is a repetitive matrix multiply program. The memory access pattern is generated using a time window of length 5000 (Figure 4a).

This program exhibits more complex memory access patterns since each processor alternates its accesses to the memory modules in a less uniform fashion than the temperature propagation program. The plot of the time averaged number of IN faults incurred while routing the memory accesses is shown in Figure 4b. This shows that the TDNN predictions aid the SSR in decreasing the number of faults after several pattern repetitions.

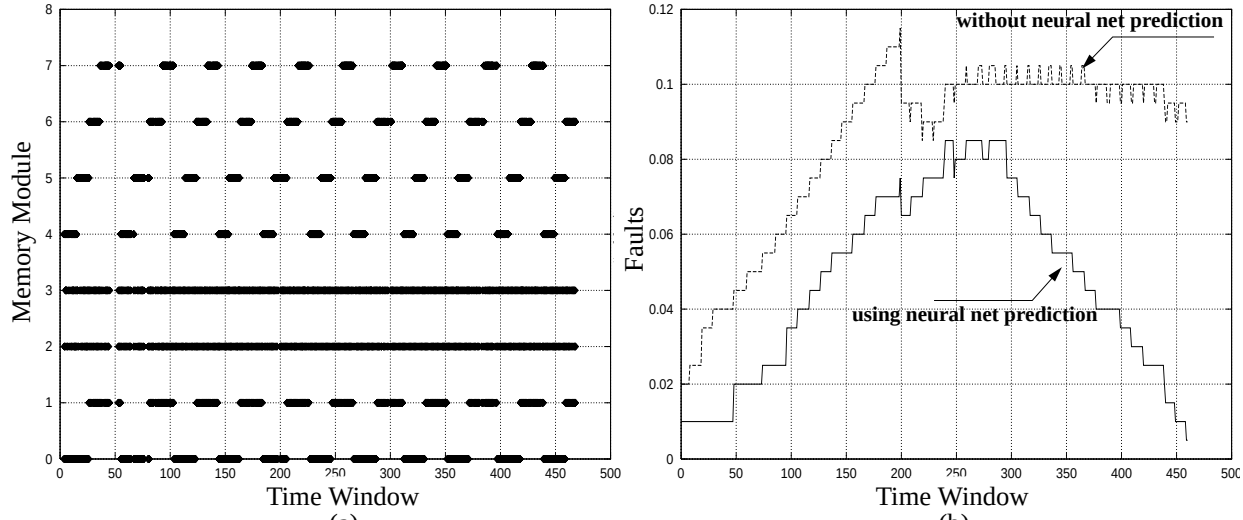

Fig. 4: (a) The memory access pattern of the matrix multiply program. (b) The number of network faults incurred with and without the TDNN predictions (time averaged).

4.3 Fast Fourier Transform

For the third experiment, the memory access pattern is generated from a repetitive 1D Fast Fourier Transform (FFT) program. In this example, a window of size 1000 time units suffices to emphasize the changes while generating the memory access pattern (Figure 5a). The number of faults that occur during the routing are time averaged and plotted vs. time in Figure 5b which shows that the neural network is also capable of learning and predicting the memory access pattern of the FFT.

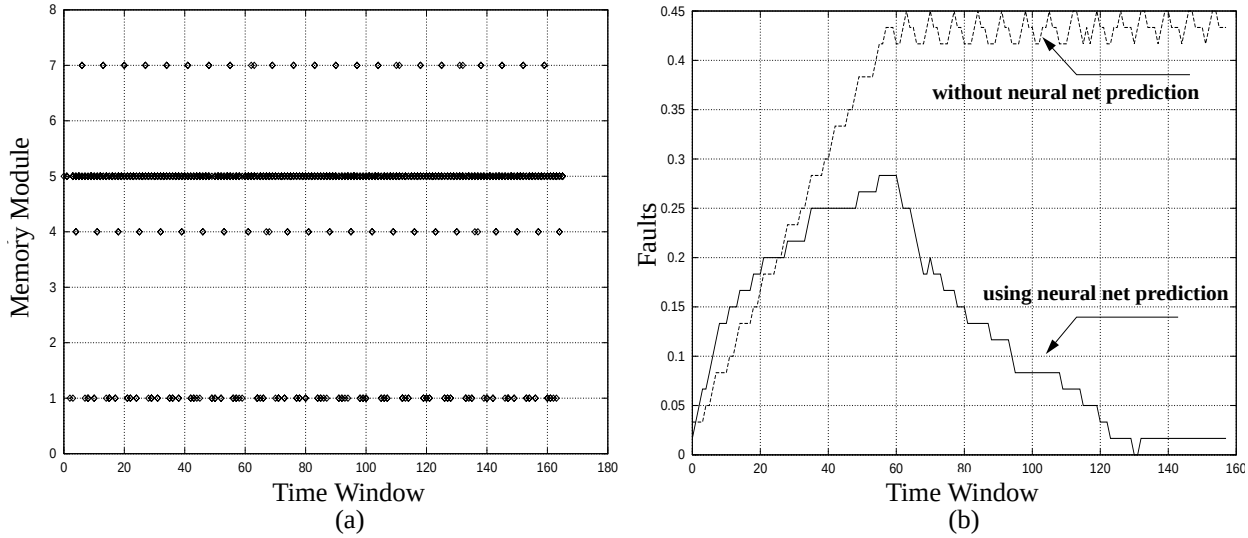

Fig. 5: (a) The memory access pattern of the FFT program. (b) The number of network faults incurred with and without the TDNN predictions (time averaged).

5 Conclusions

Large scale shared memory multiprocessors need reconfigurable interconnection networks (INs). However, these INs suffer a high overhead due to control latency. To reduce control latency, a time-delay neural network was trained to learn and predict repetitive memory access patterns for three applications, *temperature propagation*, *matrix multiply* and *Fast Fourier Transform*. The predictions were used by a state sequence routing control algorithm to reduce control latency by providing needed paths before they were requested.

The experiments show that coupling state sequence routing with the time delay neural network (TDNN) prediction technique reduces the number of faults incurred by the state sequence router. The cumulative number of faults is reduced by a factor of 2.14 for the temperature propagation program, 2.18 for the matrix multiply program and 3.28 for the FFT program. Furthermore, we show that the faults decrease with time, and we speculate that if the programs execute for longer periods than in our experiments, the cumulative number of faults will be reduced by a much larger factor. Reducing the number of faults reduces the control latency toward the limit of *k/2*, half the sequence length.

The ability of the TDNN to learn the memory access patterns of these parallel programs is based on two observations: the parallel programs exhibit some underlying characteristic for repetitiveness in their memory accesses; and, the changes in the memory access patterns are emphasized by using windowing techniques which compress the overall access pattern. The windowing process facilitates the learning of the memory access pattern, but the predictions are made at a granularity of the window size. Nevertheless, the state sequence router can use the TDNN predictions to do anticipatory reconfiguration of the IN and thereby satisfy the forthcoming memory accesses. Even if the predictions are made many time steps in the future, the state sequence router has the ability to store early predictions until their actual use. Therefore, the prediction of future memory accesses is performed successfully by the TDNN which reduces the communication latency for the applications tested.

5.1 Future Work

Our immediate plan is to compare the performance of the TDNN predictor to that of an on-line linear predictor to determine if neural network predictors are really needed. We also plan to test the performance of a first order Markov predictor. Other machine learning algorithms could also prove attractive.

We must address how these prediction methods effect actual performance of real multiprocessors. Can these prediction methods be implemented in hardware and can their results be effectively used? Finally, we would like to investigate the applicability of time series prediction techniques to the general problem of latency hiding at all levels of the memory hierarchy.

Acknowledgments

S. P. Levitan and D. M. Chiarulli would like to acknowledge support from AFOSR Grant F-49620-93-1-0023 for work done at the University of Pittsburgh.

References

[1] Bigrigg, M. “Personal Communication,” 1992.

[2] Chiarulli, D. M., Levitan, S. P., Melhem, R. G., Qiao, C., “Locality Based Control Algorithms for Reconfigurable Interconnection Networks,” *Applied Optics*, vol. 33, pp. 1528-1537, 1994.

[3] Johnson, K. L., “The Impact of Communication Locality on Large-Scale Multiprocessor Performance”, *Computer Architecture News*, vol. 20, pp 392-402, 1992.

[4] Lang, K. J., Waibel, A. H., Hinton, G. E., “A time-delay neural network architecture for isolated word recognition,” *Neural Networks*, vol. 3, pp. 23-44, 1990.

[5] Maggini, M., “Personal Communication,” 1994.

[6] Sakr, M. F., *Predicting Multiprocessor Communication Patterns with Neural Networks*, M.S. Thesis, Dept. of EE, University of Pittsburgh, 1995.

[7] Sakr, M. F., Levitan S. P., Giles C. L., Horne B. G., Maggini M., Chiarulli D. M., *Predictive Control of Opto-Electronic Reconfigurable Interconnection Networks using Neural Networks*, Proceedings of the Second International Conference on Massively Parallel Processing Using Optical Interconnections, 1995.

[8] Siegel, H. J. (1990) *Interconnection Networks for Large-Scale Parallel Processing*, McGraw-Hill, NY.

[9] Weigend, A. S., Gershenfeld, N. A. (1993) *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley.