Performance of On-Line Learning Methods in Predicting Multiprocessor Memory Access Patterns

Majd F. Sakr[1,2], Steven P. Levitan[2], Donald M. Chiarulli[3], Bill G. Horne[1], C. Lee Giles[1,4]

[1] *NEC Research Institute, 4 Independence Way, Princeton NJ 08540*
[2] *University of Pittsburgh, Electrical Engineering Department, Pittsburgh PA 15261*
[3] *University of Pittsburgh, Computer Science Department, Pittsburgh PA 15260*
[4] *UMIACS, University of Maryland, College Park, MD 20742*

Technical Report
UMIACS-TR-96-59 and CS-TR-3676
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

Abstract

Shared memory multiprocessors require reconfigurable interconnection networks (INs) for scalability. These INs are reconfigured by an IN control unit. However, these INs are often plagued by undesirable reconfiguration time that is primarily due to control latency, the amount of time delay that the control unit takes to decide on a desired new IN configuration. To reduce control latency, a trainable prediction unit (PU) was devised and added to the IN controller. The PU's job is to anticipate and reduce control configuration time, the major component of the control latency. Three different on-line prediction techniques were tested to learn and predict repetitive memory access patterns for three typical parallel processing applications, the *2-D relaxation algorithm*, *matrix multiply* and *Fast Fourier Transform*. The predictions were then used by a routing control algorithm to reduce control latency by configuring the IN to provide needed memory access paths before they were requested. Three prediction techniques were used and tested: 1). a Markov predictor, 2). a linear predictor and 3). a time delay neural network (TDNN) predictor. As expected, different predictors performed best on different applications, however, the TDNN produced the best overall results.

Keywords: Prediction; Learning; Multiprocessors; Memory; Markov Predictor; Linear Predictor; Time Delay Neural Network

1 Introduction

Large scale multiprocessor systems will require low-cost, highly-scalable, and dynamically reconfigurable interconnection networks (INs) [Siegel90]. Such INs offer a limited number of communication channels that are configured on demand to satisfy required processor-memory accesses. In this demand driven environment, a processor accessing a memory module makes a request to an IN controller to establish a path (reconfigure the IN) that satisfies the processor's request. The controller is used to optimize the required IN configuration based on the set of current 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 (Figure 1). Launch time can be reduced by using high bandwidth opto-electronic INs, and fly time is relatively insignificant in such an environment since the end-to-end distances are relatively short. Therefore, control time dominates the communication latency. However, in a multiprocessor system executing a parallel scientific application, the memory-access requests made by the processors follow a repetitive pattern based on the application. Compilers can analyze an application and attempt to predict its access patterns [Gornish90], but often the pattern is dynamic and thus hard to predict. The goal of this work is to employ a technique that learns these patterns on-line, predicts the processor requests, and performs IN configuration prior to the requests being issued, thus hiding the control latency. The effect is a significant reduction in the communications latency for multiprocessor systems.

Learning methods have been applied in various areas of computing and communication systems. For instance, neural networks have been applied to learn both network topology and traffic patterns for routing and control of communication networks [Fritsch91, Jensen90, Thomopoulos91]. Recent work on using neurocomputing in high speed communication networks was the subject of a special issue of *Communications* [Habib95]. Other applications of neural networks are for the control of switching elements of a multistage interconnection network for parallel computers [Funabiki93, Giles95] and for learning the structure of interconnection networks [Goudreau95]. For multicomputer systems, genetic algorithms have been applied as a distributed task scheduling technique [Wang95]. Solutions to the problem of mapping parallel programs onto multicomputer systems to provide load balancing and minimize interprocessor communication have been proposed using genetic algorithms [Seredynski94] and self organizing maps [Dormans95] as well as variants of the Growing Cell Structures network [Tumuluri96]. In uniprocessor environments, Stigal et. al. [Stigal91] propose a neural network cache replacement algorithm. Their technique predicts which cache block will be accessed furthest in the future and therefore should be replaced, thus lowering the cache miss rate. In general, the literature on machine learning in computing and communication systems has focused on how these techniques can be used to identify patterns of communication in order to optimize the control of these systems.

The focus of this work is to study how three on-line learning methods perform at predicting processor-memory access patterns in a multiprocessor environment. We use a Markov predictor, a linear predictor and a time-delay neural network (TDNN) to learn and predict the memory access patterns of three parallelized scientific applications: a *2-D relaxation algorithm*, a *matrix multiply*, and a *1-D FFT*. The next section presents the environment of our experiment where we describe a shared memory multiprocessor model employing prediction units (PUs). In section 3, we describe the three prediction methods used and in section 4 we present experimental results of the predictors. The final section interprets our results and discusses future directions of research.

2 Multiprocessor Model

Shared memory parallel computers are commonly referred to as *multiprocessor* systems [Bell85, Kumar94]. Our shared memory multiprocessor (SMM) system consists of 8 processors (P0-P7), 32 memory modules (M0-M31), a reconfigurable IN and an IN controller (Figure 2). This SMM model uses a state-sequence router [Chiarulli94] as the reconfigurable interconnection network controller. In addition, we use a SMM simulator which allows us to record the memory access traces of parallel applications.

In such systems with N processors and K memory modules, the reconfigurable IN can be configured to achieve any of the $N \times K$ possible *paths* between a processor and a memory module; however, it can only provide a subset of these paths at any given time. A group of compatible (non-blocking) paths are called an IN *configuration* or a *state*. Because of contention for paths, the IN must be dynamically reconfigured to satisfy the set of current processor-memory accesses. This SMM model employs an IN control system based on the *state sequence routing* (SSR) paradigm [Chiarulli94] which takes advantage of the locality characteristics exhibited in memory access patterns [Johnson92] and reconfigures the network through a fixed set of configurations in a repetitive manner. The IN controller, used for state sequence routing, consists of a *state generator* which is controlled by a *state transformer*. The state generator maintains a collection of configurations, called a *state sequence* and periodically reconfigures the IN with a new configuration from the set. Specifically, the state sequence is maintained in a cyclic shift register of length k as shown in Figure 2. With each register shift, an IN configuration is broadcast to the processors, memory modules, and switching elements of the IN. The state sequence router exploits the memory access locality inherent in these patterns by re-using the sequence of states. The state transformer is responsible for determining the set of configurations contained within the state generator based on processor requests. A processor that needs to access a memory module issues a *fault* (or request) to the state transformer only if the current state sequence does not already include the required path to a memory module. In response, the state transformer adds the required path to the state sequence by removing the least recently used path.

Using SSR the average control latency, L, incurred by each access can be shown to be:

$$L = (1-p)\frac{k}{2} + p(k+f) \tag{1}$$

where p is the probability of a fault, k is the sequence length, and f is the fault service time. If a processor needs a path and it exists in the state sequence, there is no fault issued and the latency is just the time for the path to come around in the sequence which on an average is $k/2$. However if the path does not exist after k broadcasts, the processor issues a fault which must be serviced before the memory access can occur. The SSR based IN controller needs only to establish the initial paths and respond to the changes in the memory access pattern; it is not required to respond to individual memory access requests.

Our goal is to employ 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 will transform the state sequence to include the soon-to-be-needed path, avoiding the latency incurred by the fault. As shown in Figure 2, a prediction unit (PU) is used to learn the access pattern of each processor. The predictions made by the PU are used as hints by the SSR while routing the memory accesses. Since, processor-memory access patterns change dynamically and thus can be modeled as a time series, for this preliminary investigation, we chose to study three simple on-line time series prediction methods: a Markov predictor, a linear predictor and a TDNN [Sakr95a].

3 Prediction Method Experiments

To evaluate the performance of various prediction methods, we test how well each technique can predict the next memory access pattern as the SMM executes three typical parallelized scientific applications. The first application is a parallel (32×32) 2-D grid-based temperature propagation/relaxation algorithm; the second application is a repetitive (24×12 • 12×24) matrix multiply program; the third is the memory access pattern generated from a repetitive 1D Fast Fourier Transform (FFT) of a 16 sample vector.

Each experiment consists of three distinct phases: First, using the shared memory multiprocessor (SMM) simulator, we generate the memory accesses of a parallel program assuming fixed latency in the IN and memory modules. Using the raw memory accesses generated by the SMM simulator, we extract the sequence of memory accesses of a single processor. This memory access is represented differently depending on the predictor used. For each experiment we use the 32 memory module access pattern of a single processor, these patterns are shown in Figures 5a, 6a, and 7a. The applications are symmetrically partitioned to execute the same code on all processors while each processor uses different parts of the data [Sakr95b]. Hence, the access patterns of all other processors are very similar to the one used. Second, we use the processor's memory access patterns as input to the PU to perform on-line training and one-step ahead prediction of the next memory access. Third, we evaluate the predictions by simulating the multiprocessor behavior with and without the predictions and monitor the number of faults incurred. For each of the experiments we use a relatively short state sequence length (k). As can be seen from Equation 1, the optimum sequence length, k, is a trade off between increasing k to reduce faults, and keeping k small to reduce waiting time. The values of k were chosen to minimize the faults for these applications, for the non-predictive case. We tested using the best 1, 2, 3 and 4 predictions as hints to the SSR controller. The three prediction methods tested are considered appropriate for this dynamic system since the training and prediction is performed on-line.

3.1 Markov Predictor

There are many ways one could consider using a Markov predictor [Isaacson76]. We consider both a first and second order predictor which calculates the conditional probability p of accessing memory module M_i given processor P_k has just accessed memory module M_j, i.e. $p(M_i|M_j;P_k)$.

Similarly, for the second order, we calculate $p(M_i|M_j,M_q;P_k)$ where the conditional probability is conditioned on processor P_k previously accessing memory module M_q, then M_j. Since in this model we use one PU per processor, the input of the Markov prediction unit is the temporal sequence depicting the memory access pattern of a processor (Figure 3). The probabilities are stored in a probability transition matrix. For the first order predictor, probability p_{ij} corresponds to the probability of accessing memory module i if the processor is currently accessing memory module j. Similarly for the second order predictor, $p_{i(jq)}$ corresponds to the probability of accessing memory module i if the processor is currently accessing memory module j after completing an access to memory module q. Each entry in the transition matrix is updated and normalized on-line as the application execution proceeds. For example, in the first order Markov predictor of processor P_0, the probability of processor P_0 going from M_1 to M_2 at time step t is calculated as the number of transitions P_0 has performed from M_1 to M_2 divided by the total number of times P_0 has accessed M_1 from time 0 to time t. The number of parameters needed for the first order Markov predictor is 1024 probabilities while the number of parameters for the second order Markov is 32K probabilities. However, both first and second order predictors update 32 probabilities on-line with every access since the next access could go to one of 32 memory modules. At any given time, the non-zero probabilities are used as the predictions given as hints to the state sequence router. However, the number of non-zero probabilities could be up to 32, therefore, a fixed number of the most likely predictions is specified. We tested using the highest 1, 2, 3, and 4 probabilities as predictions. Table 1 shows the percentage of faults eliminated by the first and second order Markov predictors while varying the number of predictions used as hints to the state sequence router and the size of the state sequence (k). The negative values in all tables indicate that incorrect predictions are inserted in the state sequence, impairing performance. The shadowed cells are the parameter settings that produce the best results for each access pattern. These results are discussed in more detail in the results section and depicted in Figures 5c, 6c, and 7c where we compare the performance of the Markov predictor to that of the Linear and TDNN.

3.2 Linear Predictor

For the Linear PU, the input data is transformed from a processor's raw 32 memory module traces into a sequence of 32 bit binary vectors [Sakr95b]. The i^{th} component of the binary vector is set to 1 when an access to the i^{th} memory module takes place. All other values in that vector are set to zero, this data encoding is shown in Figure 4.

For each value in the binary symbol vector we use a next step linear predictor which attempts to predict the next access based on a linear combination of all the values in the vector and their history. Since there are 32 memory modules (1×32 access vector) in the system tested, we use 32 linear predictors that predict the next access vector in parallel. In order to compare the results of this predictor with that of the TDNN we use one bias weight for each output value, hence the Linear predictor is actually an affine predictor [HechtN91]:

$$\hat{x}_i(t+1) = \sum_{k=0}^{l}\sum_{j=1}^{32} w_{ijk}x_j(t-k) + w_{i0} \qquad i = 1,2,\ldots 32;\ l = 1,5,10 \tag{2}$$

where $\bar{x}$ is a binary vector of dimension 32, x_i denotes the i^{th} component and $\hat{x}$ is the prediction. Since we are implementing one-step-ahead prediction, the Linear predictor takes as input the current binary vector and the past l history vectors and attempts to predict the vector $\bar{x}$ at the next time step (Equation 2). We tested the performance of the Linear predictor using l = 1, 5, and 10 past vectors. Therefore, the number of inputs for the three Linear predictors tested are 64, 192, 352 ($32 \times (l + 1)$) and the number of coefficients (weights) to update at each time step is 2080, 6176, 11296 respectively. The learning algorithm is a simple on-line gradient descent algorithm using the following adaptive *learning rate* [Maggini94], starting value is set to 0.01:

```
if (present error - previous error > previous error × allowed % increment [default 10%]){
        reduce learning rate (multiply by a decrease factor < 1 [default 0.5])
        and move back in the weight space to the previous point}
else {
        keep the updated weights and increase the learning rate
        (multiply by an increase factor > 1 [default 1.1])}
```

The algorithm is performed on-line, so we make only one pass through the data. Furthermore, the outputs (predictions) with values > 0.5 of which the largest values are selected as the predictions which are passed along to the state sequence router as hints. We tested using 1, 2, 3 or 4 predictions as hints to the SSR controller. Table 2 shows the percentage of faults eliminated for all the parameters explored. Again, the shadowed cells are the parameter settings that produce the best performance of the Linear predictor, these results are depicted in Figures 5d, 6d, and 7d.

3.3 Time Delay Neural Network

The data encoding of the memory accesses for the TDNN is the same as that of the Linear predictor which is shown in Figure 4. Again, since we are implementing one-step-ahead prediction, the TDNN takes as input the current binary vector and attempts to predict the access vector at the next time step [Sakr96]. Therefore there are 32 inputs and 32 outputs for the network. For each input, we experiment with a tapped delay line of length 1, 5 or 10. The total number of inputs to the MLP section of the TDNN is 64, 192, 352 derived from (32 × (1 input + # taps)). We tested the performance of the TDNN using a single hidden layer of size 10, 20 and 30 neurons. Every output node has an additional bias weight, we use tapped delay lines of sizes 1, 5 and 10. This gives 1002, 2282, 3882 total weights for the TDNN with 10 nodes in the hidden layer; 1972, 4532, 7732 total weights for the TDNN with 20 nodes in the hidden layer; and 2942, 6782, 11582 total weights for the TDNN with 30 nodes in the hidden layer. Nodes in the hidden layer use a hyperbolic tangent activation function, while nodes in the output layer are affine. All of the weights were initialized uniformly in the range [-1/ϕ, 1/ϕ], where ϕ is the number of connections that enter a node (fan in). The learning algorithm is a simple on-line gradient descent algorithm using the same adaptive learning rate used for the Linear predictor. Since the training and prediction is performed on-line, we make only one pass through the data. Many prediction interpretations could be used; we found that best performance was achieved if the output neurons with the largest values are selected as predictions. We tested using 1, 2, 3 or 4 output neurons with the highest values as prediction hints to the SSR controller. Tables 3, 4 and 5 show the percentage of faults eliminated for the parameters explored for the 2D relaxation algorithm, matrix multiply and FFT respectively. The percentage of eliminated faults is the average of five simulations using different seeds to initialize the weights, the variance is given in parenthesis. The best performance of the TDNN is shown in the shadowed cells, plots portraying the system performance using these parameter settings are depicted in Figures 5e, 6e, and 7e.

4 Results

In this section we discuss in detail the performance of the three prediction units tested for the three applications implemented on our SMM model. In order to compare the performance of the prediction units, for each application we plot the memory access pattern followed by fault plots. First we show the characteristic access pattern of each of the applications in Figures 5a, 6a, and 7a. Then the network faults incurred for the non-predictive case (Figures 5b, 6b, and 7b) followed by the network faults incurred by the system using the PUs (Figures 5c-e, 6c-e, and 7c-e).

4.1 2-D Relaxation

Figure 5a shows 8697 access vectors which depict the access behavior of the 2-D Relaxation algorithm, the large discontinuity in the pattern is a no-memory-access period which is a characteristic of the algorithm. The access patterns exhibit a stair-like behavior, where each stair discontinuity reflects a change in the memory module access. For this access pattern the Markov predictor (Table 1) performed the best of the three prediction units tested. The second order Markov predictor shows improved performance over the first order only for the 1 prediction case. In general, increasing the number of predictions used by the state sequence router enhanced performance while increasing the size of the state sequence (k) does not for this particular application. For the Linear predictor (Table 2), increasing the history or the number of predictions used as hints does not enhance performance. On the other hand, increasing k helps increase the total number of network faults eliminated. We tested many TDNN configurations (Table 3), the performance of the TDNN in predicting this pattern relied heavily on the number of nodes in the hidden layer. From Table 3, we can see that increasing the history used (tapped-delay line) does not improve performance as much as increasing the size of the hidden layer. Using a large k is also crucial in fault elimination for this pattern. Figure 5b plots the network faults incurred as impulses for the non-predictive case. The other fault plots show the best performance of the on-line predictors for the 2-D relaxation algorithm. The Markov predictor performs best for this pattern since the total number of non-zero probabilities is small (three), and using the top three probabilities is enough to predict almost perfectly and eliminate all faults (Figure 5c). The Linear predictor needs a few more training iterations before its predictions start to greatly reduce the number of faults

(Figure 5d). The TDNN with a hidden layer of 30 nodes and single node in each tapped delay line produced its best result, shown in Figure 5e.

4.2 Matrix Multiply

The matrix multiply application exhibits a more complex pattern since each processor accesses the memory modules in a less uniform fashion than the 2-D relaxation algorithm. Figure 6a shows the 11561 vector access pattern. Since this application exhibits a complex access pattern the first and second order Markov predictors cannot capture and predict the access pattern correctly. Increasing the number of predictions or *k* does not enhance overall performance. The performance of the Linear predictor (Table 2) is similar to that of the Markov predictor for this application. Varying the history, or *k*, or the number of predictions does not improve performance. On the other hand, the TDNN (Table 4) produces marginally better results. The TDNN performs best with a hidden layer of 10 nodes, using 2 predictions and a large *k*. However, increasing the size of the hidden layer hinders the performance of the TDNN. Figures 6b-e show the best performance of all three prediction units for this access pattern. In this case, the Markov predictor performs poorly since the probabilities are of equal values which increases the number of wrong predictions. The linear predictor is not able to find a linear combination of the past accesses to predict the next access well. However, the TDNN still achieves a moderate reduction in the number of faults.

4.3 Fast Fourier Transform

The FFT application produces the memory access pattern shown in Figure 7a. The Markov predictor is capable of capturing and predicting this pattern thus eliminating almost all the faults incurred by the state sequence router. The second order Markov captures the pattern of access thus producing better results compared to the first order Markov. Increasing the number of predictions used is essential for good performance while increasing *k* does not affect the percentage of fault elimination for both the first and second order Markov predictors. The Linear predictor produces its best results when using the least history, the least number of predictions and a *k* of size 6 (see Table 2). Increasing the history used or the number of predictions does not boost performance. The TDNN is capable of learning and predicting the access pattern of the FFT. As shown in Table 5, a hidden layer consisting of 10 nodes provides better performance than the TDNN with a larger hidden layer. A number of predictions of either 2 or 3 with a *k* of size 6 or 7 are needed for good performance. Figures 7b-e show the best results from the shaded cells of tables 1, 2, and 5. Since the FFT algorithm exhibits a simple access pattern, the Markov predictor is able to capture the pattern and predict well using the top four predictions. The Linear predictor is capable of eliminating some of the network faults. The TDNN (Table 5) performs better than the Linear predictor for this pattern but not better than the Markov.

5 Summary and Conclusions

Completely connected interconnection networks (INs) are not feasible in large scale multiprocessor systems because of their high complexity and soaring cost. Accordingly, we utilize less expensive reconfigurable interconnection networks that scale well but suffer from high overhead due to control latency. Control latency is the time delay incurred by the network controller to determine a new desired IN configuration and to physically establish the paths in the network. Each reconfiguration request (network fault) is triggered when the current network configuration fails to satisfy a processor's memory access. These requests are performed on a demand driven basis. However, memory access patterns of multiprocessor systems executing parallel scientific applications exhibit a lot of repetitiveness due to loops which are a characteristic of such applications. Hence, in this work we study how three learning methods perform at learning and predicting these access patterns on-line. Correct prediction of the access patterns allows anticipatory reconfiguration of the IN and thereby satisfy the forthcoming memory accesses preventing a network fault. Thus, the average control latency L is hidden and consequently overall communication latency is reduced.

The three on-line prediction methods tested are: a first and a second order Markov predictor; a linear prediction method; and a time delay neural network (TDNN). We train the prediction methods using the access patterns of three parallel scientific applications: a 2D relaxation algorithm; a matrix multiply; and a Fast Fourier Transform (FFT). The multiprocessor model used is an 8 processor 32 memory module shared memory system with a state sequence router as the reconfigurable interconnection controller.

The experiments show that coupling state sequence routing with different types of on-line prediction methods can decrease the number of memory access faults across different applications with some methods being more effective than others. The best results of the prediction methods for the access patterns tested are as follows. For the 2D relaxation algorithm the first order Markov predictor eliminates 95% of the faults; the Linear predictor prevents 95% of the faults; and the TDNN removes 71% of the faults. While for the matrix multiply: the second order Markov predictor eliminates 6% of the faults; the linear predictor removes 1% of the faults; and the TDNN prevents 30% of the network faults from taking place. Finally, using the access patterns of the FFT: the second order Markov predictor prevents 95% of the network faults; the linear predictor eliminates 34% of the faults; and the TDNN removes 45% of the network faults. As expected, different predictors perform best on different applications. From visual inspection of the access patterns (shown in Figures 5a, 6a, 7a), one could say that the access patterns of different applications vary in complexity from the 2-D relaxation being the most simple to the matrix multiply the most complex. All prediction methods perform well on the applications with the simple access patterns. On the other hand, for very complex patterns, the Markov and Linear prediction methods perform very poorly and TDNN performs the best of the three.

Given the multiprocessing environment, different applications exhibit very different patterns and a technique that will predict well across patterns is more appealing than a technique that performs best for specific patterns. Thus, we hypothesize that the TDNN has the best chance of adapting to different memory access patterns from the variety of real applications. However, it could be feasible to use all prediction methods in a mixture of experts model [Jordan94] and use the best predictor available.

Our future work will address more realistic simulation of the multiprocessor environment, such as the effects of incorporating runtime delays due to memory and network contention in the memory access patterns and how these prediction methods affect actual performance. We plan to test the performance of different machine learning techniques and other prediction methods. Also, it would be interesting to investigate the applicability of prediction techniques to the general problem of latency hiding at all levels of the memory hierarchy. Another open question is how will these prediction methods be efficiently implemented in hardware and their results effectively used. For example how and what is the effect of memory fault prediction in the actual speedup of applications on multiprocessor?

Acknowledgments

The authors would like to thank Marco Maggini for providing the on-line neural network simulator. 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

[Bell85] C.G. Bell, "Multis: A new class of multiprocessor computers," Science, vol. 228, pp. 462-467, 1985.

[Chiarulli94] D.M. Chiarulli, S.P. Levitan, R.G. Melhem, C. Qiao, "Locality Based Control Algorithms for Reconfigurable Interconnection Networks," *Applied Optics*, vol. 33, pp. 1528-1537, 1994.

[Dormans95] M. Dormans, H.-U. Heiss, "Partitioning and Mapping of Large FEM-Graphs by Self-Organization," *Proceedings Euromicro Workshop on Parallel and Distributed Processing*, San Romeo, Italy, pp. 227-235, 1995.

[Fritsch91] T. Fritsch, W. Mandel, "Communication Network Routing Using Neural Nets-Numerical Aspects and Alternative Approaches," *IEEE International Joint Conference on Neural Networks*, pp 752-757, 1991.

[Funabiki93] N. Funabiki, Y. Takefuji, K.C. Lee, "Comparisons of Seven Neural Network Models on Traffic Control Problems in Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. 42, no. 4, pp 497-501, April 1993.

[Giles95] C.L. Giles, M.W. Goudreau, "Routing in Optical Multistage Interconnection Networks: a Neural Network Solution," *Journal of Lightwave Technology*, vol. 13, no. 6, June 1995.

[Gornish90] E.H. Gornish, E.D. Granston, A.V. Veidenbaum, "Compiler-directed Data Prefetching in Multiprocessors with Memory Hierarchies," *Proceedings of the 1990 International Conference on Supercomputing*, pp. 354-368, Sep. 1990.

[Goudreau95] M.W. Goudreau, C.L. Giles, "Using Recurrent Neural Networks to Learn the Structure of Interconnection Networks," *Neural Networks*, vol. 8, no. 5, pp. 793-804 (1995).

[Habib95] I.W. Habib, Guest editor of the Special Issue on Neurocomputing in High-Speed Networks, *IEEE Communications Magazine,* vol. 33*,* October 1995.

[HechtN91] R. Hecht-Nielsen, *Neurocomputing*, Addison Wesley, 1991.

[Isaacson76] D.L. Isaacson, R.W. Madsen, *Markov Chains Theory and Applications,* R.E. Krieger, 1976.

[Jensen90] J.E. Jensen, M.A. Eshera, S.C. Barash, "Neural Network Controller for Adaptive Routing in Survivable Communication Networks," *IEEE International Joint Conference on Neural Networks*, pp 2693-702, 1990.

[Johnson92] K.L. Johnson, "The Impact of Communication Locality on Large-Scale Multiprocessor Performance*,*" *Computer Architecture News*, vol. 20, pp 392-402, 1992.

[Jordan94] M.I. Jordan, R.A. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm," *Neural Computation*, vol. 6, pp. 181-214, 1994.

[Kumar94] V. Kumar, A. Grama, A. Gupta, G. Karypis, "Introduction to Parallel Computing," Benjamin/Cummings, CA, 1994.

[Maggini94] M. Maggini, Personal Communication, 1994.

[Sakr95a] M.F. Sakr*, "*Predicting Multiprocessor Communication Patterns with Neural Networks," *M.S. Thesis*, Electrical Engineering Department, University of Pittsburgh, 1995.

[Sakr95b] M.F. Sakr, S.P. Levitan, C.L. Giles, B.G. Horne, M. Maggini, D.M. Chiarulli, "Predictive Control of Opto-Electronic Reconfigurable Interconnection Networks using Neural Networks," *Proceedings of the Second IEEE International Conference on Massively Parallel Processing Using Optical Interconnections*, pp. 326-335, 1995.

[Sakr96] M.F. Sakr, C.L. Giles, S.P. Levitan, B.G. Horne, M. Maggini, D.M. Chiarulli, "On-Line Prediction of Multiprocessor Memory Access Patterns*,*" *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1564-1569, 1996.

[Seredynski94] F. Seredynski, "Dynamic Mapping and Load Balancing with Parallel Genetic Algorithms," *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. II, pp. 834-839, 1994.

[Siegel90] H.J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing,* McGraw-Hill, NY, 1990.

[Stigal91] P.D. Stigal, C.H. Dagli, C.F. Sen, *"*A Neural Network Cache Controller*," Intelligent Engineering Systems Through Artificial Neural Networks*, C. Dagli, S. Kumara and Y. Shin editors, pp. 561-566, ASME Press, 1991.

[Thomopoulos91] S.C.A. Thomopoulos, L. Zhang, C.D. Wann, "Neural Network Application of the Shortest Path Algorithm for Traffic Control in Communication Networks," *IEEE International Joint Conference on Neural Networks*, pp. 2693-702, 1991.

[Tumuluri96] C. Tumuluri, C.K. Mohan, A.N. Choudhary, "Unsupervised Algorithms for Learning Spatio-Temporal Correlations", Syracuse University Technical Report #SU-CIS-96-1, 1996.

[Wang95] P.C. Wang, W. Korfhage, "Process Scheduling Using Genetic Algorithms," *Proceeding of the 7th IEEE Symposium on Parallel and Distributed Processing*, pp. 638-641, 1995.

Table 1: Markov Predictor: Percentage of faults eliminated

Fixed Parameters	1st Order, 32 States				2nd Order, 1024 States			
Number of Predictions	1	2	3	4	1	2	3	4
2D Relaxation (k = 4)	0	93	96	94	35	93	96	96
2D Relaxation (k = 5)	0	93	96	96	35	93	96	96
2D Relaxation (k = 6)	0	93	96	96	35	93	96	96
2D Relaxation (k = 7)	0	93	96	96	35	93	96	96
Matrix Multiply (k = 4)	-6	-85	-75	-74	0	-3	4	4
Matrix Multiply (k = 5)	0	-45	-74	-72	3	1	-18	6
Matrix Multiply (k = 6)	0	3	-79	-71	3	-7	-60	-15
Matrix Multiply (k = 7)	0	3	-40	-76	3	5	-2	-55
FFT (k = 4)	11	32	31	94	31	53	63	95
FFT (k = 5)	11	43	53	94	31	53	74	95
FFT (k = 6)	11	43	63	85	21	63	85	95
FFT (k = 7)	14	27	54	94	42	63	85	95

The percentage of faults eliminated by the Markov PU compared to the non-predictive case. This table compares the performance of the first and second order Markov predictors while varying the number of predictions used as hints to the SSR and k (the state sequence length). The positive values indicate elimination of faults while the negative numbers are due to inserting incorrect predictions into the state sequence.

Table 2: Linear Predictor: Percentage of faults eliminated

Fixed Parameters	32 Inputs, 32 Outputs, Learning Rate = 0.01											
Tapped Delay line	1				5				10			
Number of Predictions	1	2	3	4	1	2	3	4	1	2	3	4
2D Relaxation (k = 4)	93	93	91	91	88	71	67	59	50	18	-10	-28
2D Relaxation (k = 5)	93	94	93	92	90	84	77	73	56	45	16	5
2D Relaxation (k = 6)	93	95	94	94	90	89	82	79	58	61	35	21
2D Relaxation (k = 7)	93	95	95	95	91	92	88	83	58	67	51	37
Matrix Multiply (k = 4)	-9	-12	-13	-13	-3	-7	-7	-8	-8	-20	-21	-22
Matrix Multiply (k = 5)	0	-2	-3	-3	-1	-4	-5	-5	-2	-10	-14	-14
Matrix Multiply (k = 6)	0	0	-1	-1	0	-2	-3	-4	1	-5	-10	-10
Matrix Multiply (k = 7)	0	0	0	0	0	0	-2	-3	1	0	-6	-7
FFT (k = 4)	15	14	15	13	-1	-9	-8	-15	-4	-13	-11	-20
FFT (k = 5)	28	24	24	24	3	-3	-5	-6	-1	-6	-7	-8
FFT (k = 6)	34	33	32	32	6	2	-1	-2	1	0	-2	-6
FFT (k = 7)	7	9	7	6	5	-2	-5	-8	-2	-6	-5	-10

The percentage of faults eliminated by the Linear PU compared to the non-predictive case. This table compares the performance of the Linear predictor using different history values while varying the number of predictions used as hints to the SSR and k (the state sequence length). The positive values indicate elimination of faults while the negative numbers are due to inserting incorrect predictions into the state sequence.

Table 3: TDNN: Percentage of faults eliminated for the 2D Relaxation

Tapped Delay line	1				5				10			
Number of Predictions	1	2	3	4	1	2	3	4	1	2	3	4
Hidden Layer	32 Inputs, **10 Hidden**, 32 Outputs											
***k* = 4**	0 (0)	-7 (1)	-70 (7.4)	-207 (1.2)	0 (0)	-17 (2.2)	-115 (9.4)	-189 (22)	0 (0)	-37 (13.2)	-125 (31.2)	-211 (11.6)
***k* = 5**	0 (0)	2 (0.8)	-37 (1.4)	-84 (4)	0 (0)	-1 (0.4)	-54 (1)	-127 (7.8)	0 (0)	-3 (0.2)	-64 (8.4)	-130 (10.8)
***k* = 6**	0 (0)	4 (1)	-12 (1.2)	-45 (2.6)	0 (0)	0 (0)	-16 (1.2)	-81 (4.4)	0 (0)	0 (0.2)	-22 (3)	-80 (14.8)
***k* = 7**	0 (0)	5 (1.8)	1 (0.6)	-32 (1.2)	0 (0)	0 (0)	-2 (0)	-42 (6.8)	0 (0)	0 (0.2)	-4 (0.6)	-44 (9.2)
Hidden Layer	32 Inputs, **20 Hidden**, 32 Outputs											
***k* = 4**	2 (1)	4 (2)	-79 (9.6)	-224 (11.4)	0 (0.2)	-30 (25.6)	-145 (13.8)	-212 (6.4)	0 (0.6)	-53 (60)	-155 (105)	-242 (23.8)
***k* = 5**	3 (0.8)	29 (1.4)	-38 (6.2)	-96 (1.8)	0 (0)	-4 (2.2)	-78 (3.8)	-167 (5.2)	0 (0.2)	-9 (8.19)	-85 (62.2)	-161 (50.4)
***k* = 6**	3 (0.8)	39 (3.8)	-2 (7.2)	-47 (2)	0 (0.2)	0 (0.6)	-35 (4.8)	-104 (9.2)	0 (0.2)	0 (0.8)	-43 (19.8)	-98 (15.2)
***k* = 7**	3 (0.8)	43 (5.8)	21 (2.2)	-29 (2.8)	0 (0.2)	1 (0.2)	-10 (1.2)	-64 (2.4)	1 (0.6)	1 (0.8)	-14 (6.2)	-64 (17.6)
Hidden Layer	32 Inputs, **30 Hidden**, 32 Outputs											
***k* = 4**	12 (1.2)	17 (6.2)	-81 (5.2)	-224 (32.4)	0 (1.4)	-42 (13.2)	-168 (19.8)	-216 (33.2)	-1 (0.4)	-64 (5)	-173 (15.6)	-253 (42.2)
***k* = 5**	12 (1.6)	53 (5.4)	-31 (7.8)	-99 (3)	1 (1)	-6 (4)	-91 (10)	-184 (17.2)	0 (0.2)	-15 (3)	-97 (12.6)	-180 (13.8)
***k* = 6**	12 (1.8)	67 (7)	6 (8)	-41 (5)	2 (0.6)	5 (3)	-45 (3.8)	-118 (10.4)	0 (0)	-2 (0.6)	-56 (4.4)	-116 (6.6)
***k* = 7**	12 (1.8)	71 (3.8)	33 (4)	-19 (11.4)	2 (0.6)	8 (2.8)	-13 (7.4)	-65 (7)	0 (0.4)	0 (0.2)	-23 (3)	-75 (5.6)

The percentage of faults eliminated averaged over 5 simulation runs using different seeds to initialize the weights of the TDNN. The variance is shown in parenthesis. This table compares the performance of the TDNN predictor using different tapped delay line sizes and hidden layer sizes while varying the number of predictions used as hints to the SSR and *k* (the state sequence length). The positive values indicate elimination of faults while the negative numbers are due to inserting incorrect predictions into the state sequence.

Table 4: TDNN: Percentage of faults eliminated for the Matrix Multiply

Tapped Delay line	1				5				10			
Number of Predictions	1	2	3	4	1	2	3	4	1	2	3	4
Hidden Layer	32 Inputs, **10 Hidden**, 32 Outputs											
***k* = 4**	-5 (0.4)	-90 (4)	-92 (1.2)	-86 (1.4)	-5 (0.4)	-94 (2)	-101 (3.4)	-94 (2.6)	-3 (0.6)	-39 (5.4)	-103 (2.2)	-95 (2.8)
***k* = 5**	0 (0.2)	-11 (2)	-81 (0.8)	-78 (1.2)	1 (0.2)	-24 (5.4)	-87 (2.4)	-84 (2.8)	1 (0.2)	-23 (6.8)	-86 (4)	-85 (2.8)
***k* = 6**	1 (0)	2 (0)	-29 (3)	-80 (1.4)	2 (0.4)	15 (1.4)	-39 (2.4)	-74 (0.6)	2 (0.2)	19 (2.8)	-35 (12)	-73 (5)
***k* = 7**	1 (0)	5 (0.2)	1 (1.4)	-53 (7.2)	2 (0.2)	26 (3)	12 (0.8)	-39 (3.2)	2 (0.2)	30 (2)	17 (4.4)	-34 (11.2)
Hidden Layer	32 Inputs, **20 Hidden**, 32 Outputs											
***k* = 4**	-6 (0.6)	-92 (1.4)	-92 (2)	-90 (2.6)	-5 (0.2)	-95 (3.8)	-103 (3.8)	-99 (10.8)	-5 (0.2)	-101 (8.4)	-105 (11.2)	-103 (12.8)
***k* = 5**	0 (0)	-18 (5.2)	-84 (2.2)	-79 (2.8)	1 (0.4)	-32 (1)	-94 (5.8)	-88 (5)	1 (0.4)	-35 (4.8)	-95 (17.6)	-89 (11.6)
***k* = 6**	1 (0.2)	1 (0.2)	-39 (7.2)	-83 (1.4)	1 (0.2)	8 (1.2)	-58 (4.6)	-84 (5.2)	2 (0.2)	12 (2.8)	-58 (18.4)	-84 (22)
***k* = 7**	1 (0)	4 (0.2)	-4 (1.8)	-62 (8.4)	1 (0.2)	20 (1)	-10 (2.8)	-64 (6)	2 (0.4)	26 (1.2)	-5 (6.4)	-61 (20.6)
Hidden Layer	32 Inputs, **30 Hidden**, 32 Outputs											
***k* = 4**	-7 (0.2)	-98 (1.4)	-96 (3)	-94 (1)	-5 (0.4)	-102 (4.4)	-108 (3.4)	-108 (2.4)	-6 (0.2)	-104 (11.8)	-107 (7.8)	-106 (14.6)
***k* = 5**	0 (0)	-27 (4.4)	-90 (2)	-84 (1)	0 (0.4)	-39 (4.2)	-102 (2.4)	-95 (4.4)	0 (0)	-42 (11.6)	-100 (12.4)	-92 (12)
***k* = 6**	1 (0)	0 (0.4)	-51 (3.6)	-87 (0.6)	1 (0.2)	6 (1)	-73 (0.4)	-93 (2.8)	1 (0)	7 (2.8)	-70 (15.6)	-90 (12.4)
***k* = 7**	1 (0)	4 (0.2)	-13 (2.2)	-70 (0.8)	2 (0.4)	19 (1.4)	-23 (1.6)	-79 (2.8)	2 (0.6)	23 (0.6)	-18 (10.6)	-75 (16.4)

The percentage of faults eliminated averaged over 5 simulation runs using different seeds to initialize the weights of the TDNN. The variance is shown in parenthesis. This table compares the performance of the TDNN predictor using different tapped delay line sizes and hidden layer sizes while varying the number of predictions used as hints to the SSR and *k* (the state sequence length). The positive values indicate elimination of faults while the negative numbers are due to inserting incorrect predictions into the state sequence.

Table 5: TDNN: Percentage of faults eliminated for the FFT

Tapped Delay line	1				5				10			
Number of Predictions	1	2	3	4	1	2	3	4	1	2	3	4
Hidden Layer	32 Inputs, **10 Hidden**, 32 Outputs											
***k* = 4**	5 (15.2)	-5 (11.8)	-15 (15.6)	-30 (87.6)	1 (6)	-4 (42.8)	-15 (22.8)	-17 (104)	4 (42.6)	2 (183)	-10 (146)	-20 (107)
***k* = 5**	20 (11.8)	23 (9.2)	5 (2.6)	-7 (22.6)	8 (0.4)	23 (53.8)	9 (43.6)	-2 (49.4)	11 (18.4)	30 (99.6)	12 (126)	2 (62.6)
***k* = 6**	19 (9.2)	43 (13.2)	34 (24.8)	15 (25.2)	9 (9)	42 (13.2)	33 (50.4)	16 (41.4)	12 (17)	45 (48.2)	37 (81.2)	16 (122)
***k* = 7**	3 (3.8)	27 (48.6)	44 (23.6)	24 (42)	5 (18.2)	40 (33)	40 (10.4)	18 (45.8)	9 (48.4)	41 (33.8)	43 (66.2)	17 (225)
Hidden Layer	32 Inputs, **20 Hidden**, 32 Outputs											
***k* = 4**	4 (12.6)	-7 (4.2)	-20 (30.8)	-41 (6)	0 (6.4)	-4 (28.4)	-18 (13.2)	-30 (32.6)	-1 (4.6)	-7 (17.4)	-20 (20.6)	-29 (64.2)
***k* = 5**	25 (1.4)	16 (5.6)	-6 (41.2)	-17 (10)	5 (2)	21 (62.4)	-6 (11)	-13 (36.8)	6 (5.2)	19 (13.4)	-8 (22.4)	-14 (76.2)
***k* = 6**	28 (3)	39 (17.4)	15 (51)	-6 (44.6)	7 (2.2)	45 (35.7)	14 (33.8)	-9 (19)	8 (3.2)	38 (21.8)	14 (37)	-9 (62)
***k* = 7**	6 (4)	32 (18.2)	17 (44.8)	-14 (50)	4 (3.4)	41 (45.2)	18 (25.4)	-22 (40.8)	3 (19)	34 (35.7)	15 (122)	-21 (94)
Hidden Layer	32 Inputs, **30 Hidden**, 32 Outputs											
***k* = 4**	11 (19.8)	-13 (46.8)	-27 (59.6)	-46 (45.8)	-3 (8.6)	-19 (34.2)	-33 (76.6)	-45 (67.8)	-4 (5.4)	-19 (21.2)	-32 (74)	-41 (30.2)
***k* = 5**	28 (3)	13 (59.2)	-12 (88.8)	-25 (30)	2 (0.6)	8 (35.2)	-24 (44.4)	-26 (71.6)	1 (4.4)	5 (27)	-22 (38.4)	-23 (59)
***k* = 6**	30 (10.6)	34 (16.4)	8 (104)	-15 (37.6)	7 (1.4)	30 (5)	-5 (26)	-19 (84.6)	4 (5.8)	22 (19.6)	-5 (39.6)	-20 (65.4)
***k* = 7**	8 (9)	31 (28.4)	6 (79)	-29 (90)	2 (7.4)	24 (18.8)	-7 (40.4)	-40 (60.6)	-1 (5.2)	19 (21.8)	-8 (20.8)	-36 (56.8)

The percentage of faults eliminated averaged over 5 simulation runs using different seeds to initialize the weights of the TDNN. The variance is shown in parenthesis. This table compares the performance of the TDNN predictor using different tapped delay line sizes and hidden layer sizes while varying the number of predictions used as hints to the SSR and *k* (the state sequence length). The positive values indicate elimination of faults while the negative numbers are due to inserting incorrect predictions into the state sequence.

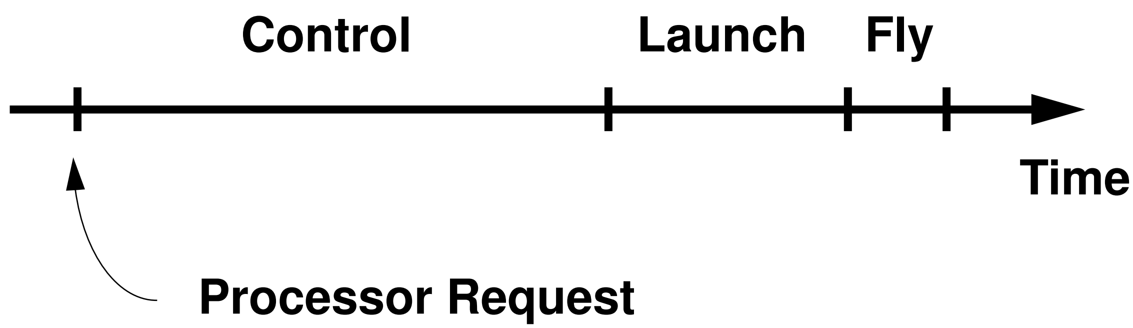

Figure 1: The three components of the end-to-end communication latency; control time, launch time and fly time. Control time dominates overall communication latency.

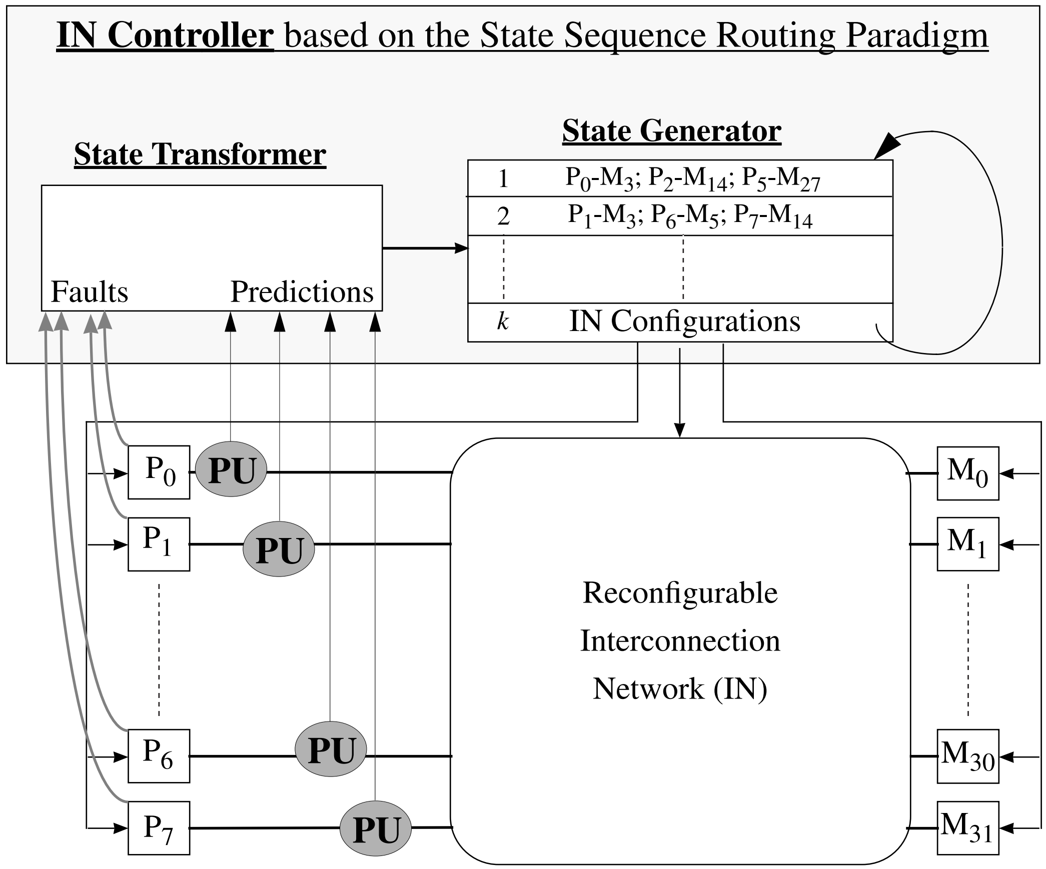

Figure 2: An 8×32 shared memory multiprocessor system employing the SSR paradigm as the IN controller and one on-line Prediction Unit (PU) per processor.

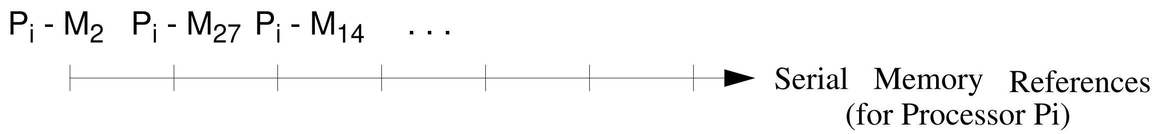

Figure 3: Data encoding for the Markov predictor.

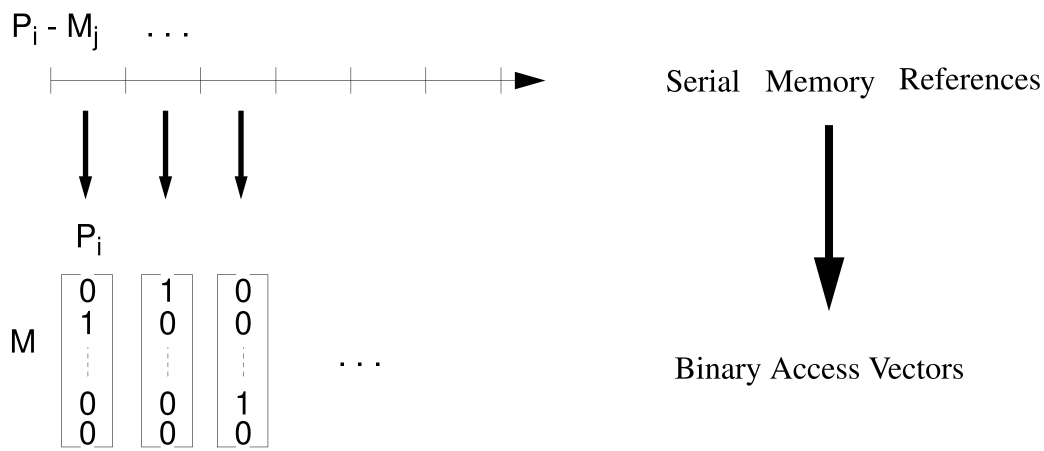

Figure 4: Data encoding for the Linear and TDNN PUs.

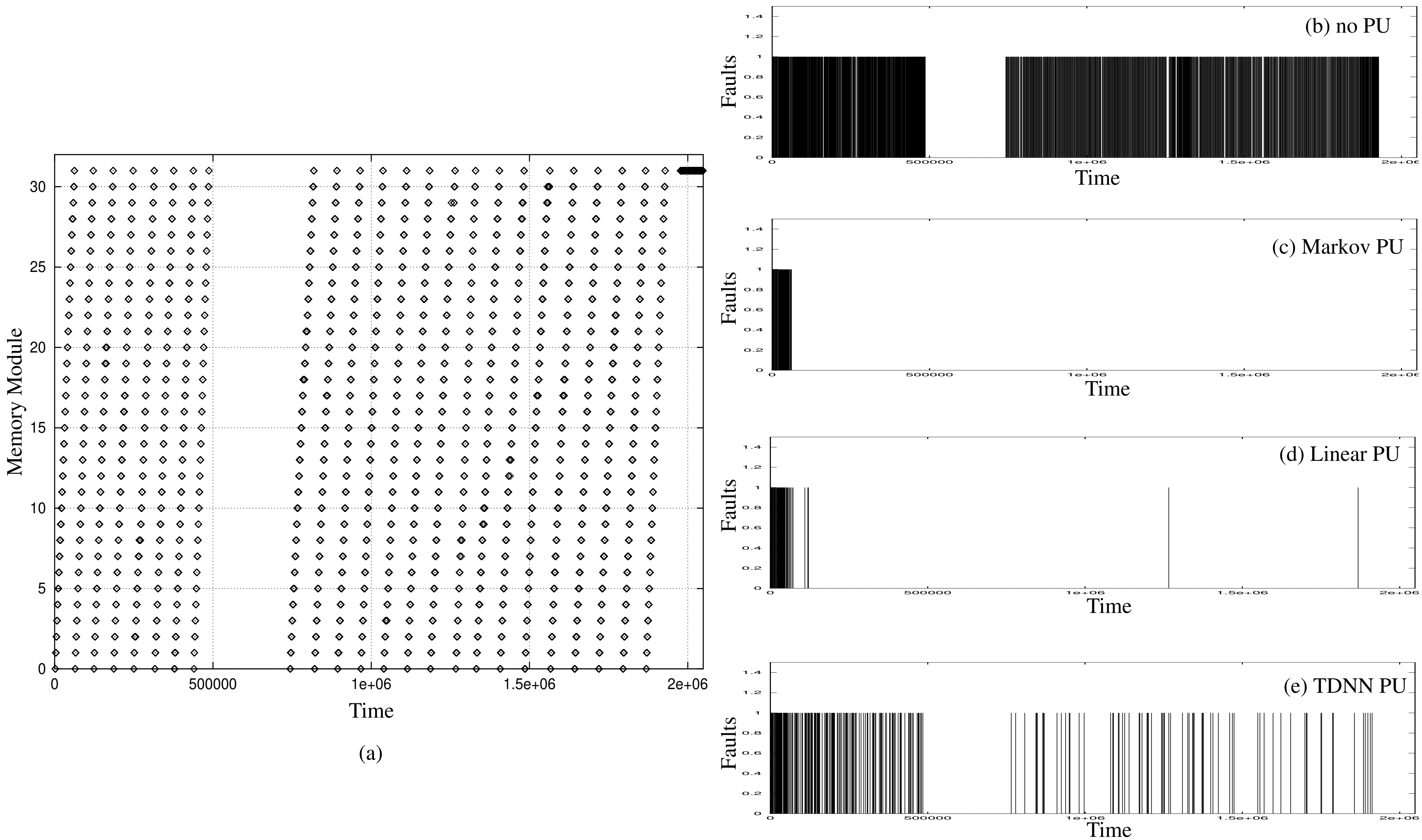

Figure 5: (a) The memory access pattern of the 2-D relaxation algorithm for Processor P_0. (b) The number of network faults incurred without predictions. (c) Number of network faults using the Markov PU. (d) Number of network faults using the Linear PU. (e) Number of network faults using the TDNN PU.

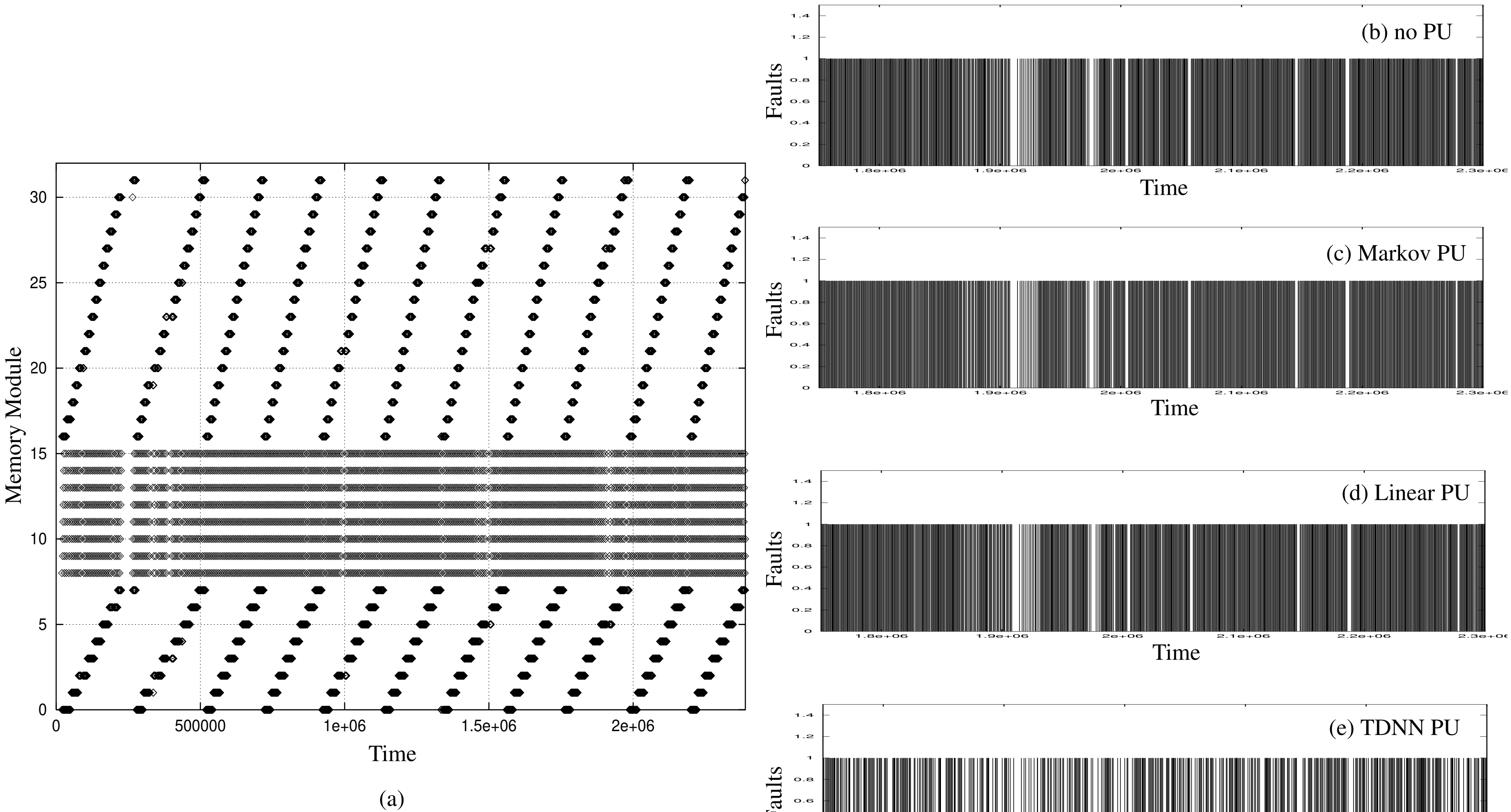

Figure 6: (a) The memory access pattern of the matrix multiply for Processor P_0. (b) The number of network faults incurred without predictions. (c) Number of network faults using the Markov PU. (d) Number of network faults using the Linear PU. (e) Number of network faults using the TDNN PU.

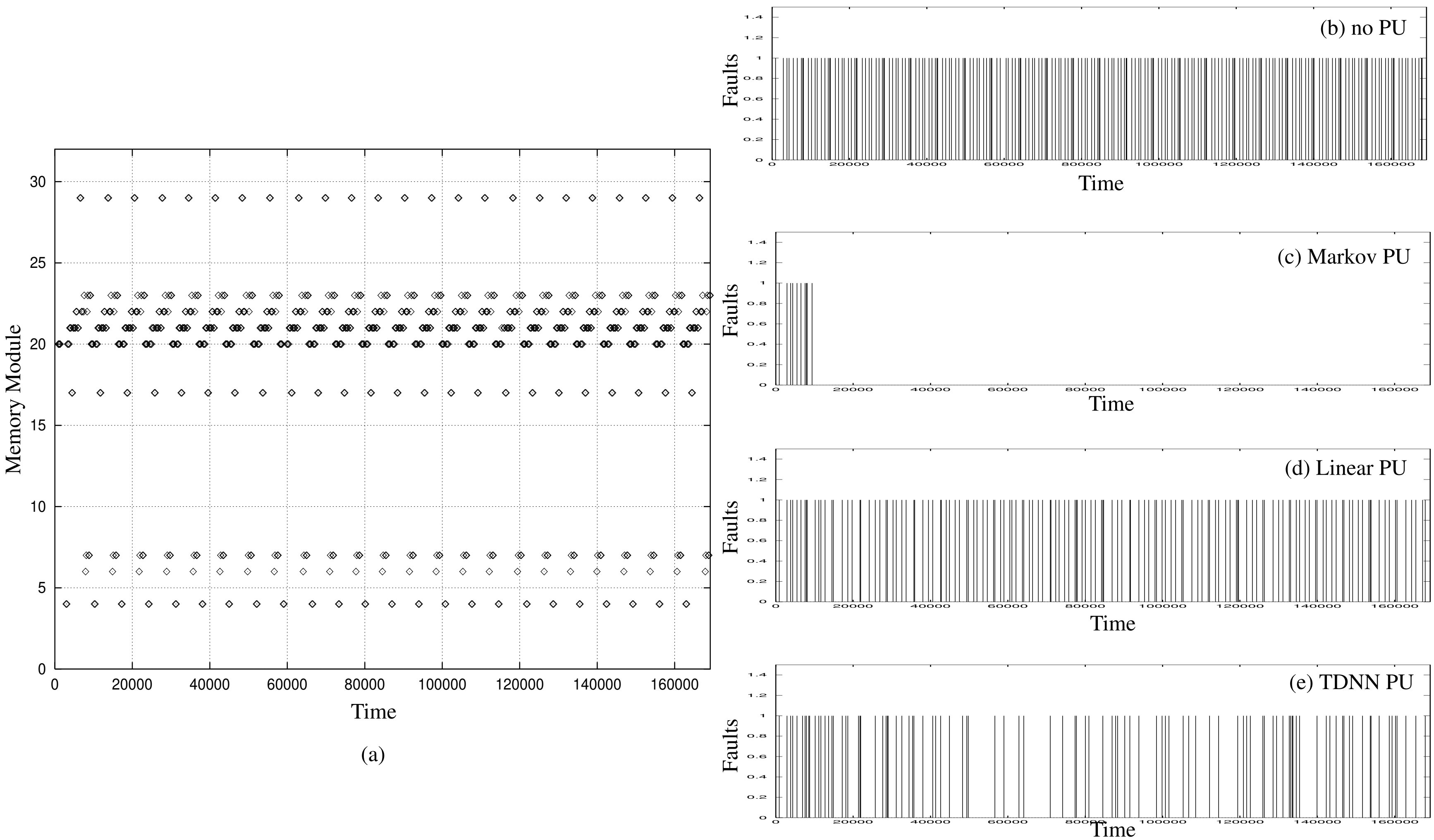

Figure 7: (a) The memory access pattern of the FFT for Processor P_5. (b) The number of network faults incurred without predictions. (c) Number of network faults using the Markov PU. (d) Number of network faults using the Linear PU. (e) Number of network faults using the TDNN PU.