Neural Comput & Applic (2013) 23:2481–2491
DOI 10.1007/s00521-012-1267-9

ORIGINAL ARTICLE

Predicting the performance measures of a message-passing multiprocessor architecture using artificial neural networks

Elrasheed Ismail Mohommoud Zayid · Mehmet Fatih Akay

Received: 14 July 2012 / Accepted: 31 October 2012 / Published online: 10 November 2012

Abstract In this paper, we develop multi-layer feed-forward artificial neural network (MFANN) models for predicting the performance measures of a message-passing multiprocessor architecture interconnected by the simultaneous optical multiprocessor exchange bus (SOME-Bus), which is a fiber-optic interconnection network. OPNET Modeler is used to simulate the SOME-Bus multiprocessor architecture and to create the training and testing datasets. The performance of the MFANN prediction models is evaluated using standard error of estimate (SEE) and multiple correlation coefficient (R). Also, the results of the MFANN models are compared with the ones obtained by generalized regression neural network (GRNN), support vector regression (SVR), and multiple linear regression (MLR). It is shown that MFANN models perform better (i.e., lower SEE and higher R) than GRNN-based, SVR-based, and MLR-based models for predicting the performance measures of a message-passing multiprocessor architecture.

Keywords Artificial neural networks · Multiprocessor architectures · Message passing · Performance evaluation

1 Introduction

Parallel computing is the simultaneous use of multiple compute processing units to solve a computational problem. Parallel computing takes hold in many areas of mainstream computing [1]. Some important domains for parallel computing today include scientific applications which model physical phenomena; engineering applications such as those in computer-aided design, digital signal processing, automobile crash simulation, and even simulations used to evaluate architectural tradeoffs; graphics and visualization applications that render scenes or volumes into images; media processing applications such as image, video and audio analysis and processing, speech and handwriting recognition; information management applications such as databases and transaction processing; optimization applications such as crew scheduling for an airline and transport control; artificial intelligence applications such as expert systems and robotics; multiprogrammed workloads; and the operating system itself, which is a particularly complex parallel application [2].

Developing parallel applications that are robust and provide good speed-up across current and future multiprocessors is a critical task and requires a tremendous amount of computational power, in addition to a deep understanding of forces driving parallel computers. Essentially, parallel computer architecture has matured to the point where it needs to be studied from a basis of engineering principles and quantitative evaluation of performance and cost. Parallel programming models are evolving apace and can truly utilize large-scale parallel computing systems. Several parallel programming models exist in common use [3], and message passing and shared-memory programming models are the most popular ones. In the message-passing model, a set of nodes use their own local memory during computation. Nodes exchange data through communications by sending and receiving messages, and data transfer usually requires cooperative operations to be performed by each process. The drawback

E. I. M. Zayid
Department of Electrical-Electronics Engineering,
Cukurova University, Adana 01330, Turkey

M. F. Akay (✉)
Department of Computer Engineering,
Cukurova University, Adana 01330, Turkey
e-mail: mfakay@cu.edu.tr

of message passing is the programmer's responsibility for determining and orchestrating all parallelism. In the shared-memory-programming model, tasks share a common address space, which they read and write asynchronously. An advantage of this model is that the notion of data "ownership" is lacking so there is no need to specify explicitly the communication of data between tasks, therefore program development can often be simplified [4–6].

The performance analysis of a multiprocessor architecture is a very crucial factor in designing message passing and shared-memory multiprocessor systems. Very often, simulation is the only feasible method because of the nature of the problem and because analytical techniques become too difficult to handle. Simulation occurs at many levels, from circuit to system, and at different degrees of detail as the design evolves. Execution-driven and trace-driven multiprocessor simulations have been extensively used to obtain a reliable and accurate prediction of the final design. One of the problems with simulation is that although these simulations can be done at a high level of abstraction, they still are extremely time consuming. There are several reasons why this is the case. First, the benchmarks that need to be simulated typically consist of several hundreds of billions of dynamically executed instructions. Second, multiple of these benchmarks need to be simulated to cover a representative set of applications. Third, the complexity of the target system reflects itself in the complexity of the simulator making the simulator at least four orders of magnitude slower than native hardware execution. Fourth, during design space exploration, all benchmarks need to be simulated multiple times to identify the optimal design for a given cost function covering performance, power, area, cost, and reliability [7].

With the objective of reducing simulation time without losing accuracy, some interesting proposals have appeared in the last years. The first one is the sampled simulation, which chooses in an intelligent way a small portion of the program trace to simulate [7]. The second one is using a reduced set of the inputs of a benchmark [7]. Finally, there is statistical modeling and simulation, which characterizes the behavior of the program and architecture with some probability distributions [8, 9]. A statistical simulation is a robust, flexible, and suitable tool in multiprocessor design, but it can still be time consuming especially when multiprocessor systems to be simulated have many parameters and these parameters have to be tested with different probability distributions or values.

There exist three studies in literature [7, 9, 10], which prove the fact that artificial intelligence techniques could be applied to predict the performance measures of a multiprocessor architecture. In [7], a broadcast-based multiprocessor architecture called the SOME-Bus employing the distributed shared-memory programming model was considered. The statistical simulation of the architecture was carried out to generate the dataset. The dataset contained the following variables: ratio of the mean message channel transfer time to the mean thread run time (T/R), probability that a block can be found in modified state (P_m), probability that a data message is due to a write miss (P_w), probability that a cache is full (P_{cf}), and probability of having an upgrade ownership request (P_{uor}). SVR was used to build prediction models for predicting average network response time (NRT), average channel waiting time (CWT), and average processor utilization (PU). It was concluded that SVR model is a promising tool for predicting the performance measures of a distributed shared-memory multiprocessor.

In [9] and [10], MFANN models were developed to predict the measures of the SOME-Bus architecture employing the message-passing protocol with (ACK's), and the hybrid message-passing protocols. The performance of the models was evaluated by calculating the mean absolute error (MAE), root mean squared error (RMSE), relative absolute error (RAE), and root relative squared error (RRSE). The results of the MFANN-based models were compared with the ones obtained by GRNN, SVR, and MLR. It is concluded that MFANN models shortens the time quite a bit for obtaining the performance measures of a message-passing multiprocessor employing the message-passing protocol with ACK's and can be used as an effective tool for this purpose.

The difference between this study and [10] is that the dataset used in [10] is a hybrid one in the sense it includes simulation results both for message passing with acknowledgments (ACK's) and without ACK's. This study uses the dataset which includes simulation results for message passing with ACK's only. This study extends our previous work [9] by including three new methods (i.e., SVR, GRNN, and MLR) for performance measures prediction.

In this paper, MFANNs have been used to predict the performance measures of the SOME-Bus architecture employing the message-passing programming model with ACK's. OPNET Modeler [11] is used to statistically simulate the message-passing SOME-Bus architecture. The input variables of the prediction model include T/R, node number, thread number, and traffic pattern. The output variables of the prediction model include average CWT, average channel utilization (CU), average NRT, average PU and average input waiting time (IWT). The performance of the prediction models have been evaluated by calculating their SEE and R values. The results are compared with the ones obtained by GRNN, SVR, and MLR. It is shown that MFANNs perform better than GRNN, SVR, and MLR analysis for predicting the performance measures of a message-passing multiprocessor architecture.

This paper is organized as follows. Section 2 gives an overview of the SOME-Bus architecture. Section 3 presents details of message-passing protocol. Section 4 summarizes the basics of MFANNs, GRNN, SVR, and MLR. Section 5 describes the simulation framework and dataset generation. Section 6 gives the MFANN prediction models, performance metrics, results, and discussion. Finally, Sect. 7 concludes the paper and outlines the future work.

2 Overview of the simultaneous optical multiprocessor exchange bus (SOME-Bus)

One implementation of an architecture which can support simultaneous multicasts from all nodes has been presented in [12]. This architecture, called the SOME-Bus, incorporates optoelectronic devices into high-performance network architecture. The unique architecture of the SOME-Bus provides for strong integration of the transmitter, receiver, and cache controller hardware to produce a highly integrated system-wide coherence mechanism. It is a low-latency, high-bandwidth, fiber-optic network which directly connects each processing node to all other nodes without contention. One of its key features is that each of N nodes has a dedicated broadcast channel which can operate at GBytes/s, depending on the configuration. Figure 1 depicts the fully connected SOME-Bus network architecture. The receiver array does not need to perform any routing, and consequently its hardware complexity is small. It contains an optical interface which performs address filtering, length monitoring, and type decoding. If a valid address is detected in the message header, the message is placed in a queue, otherwise the message is ignored. The address filter can recognize broadcast addresses in addition to recognizing the address of the home node. The receiver array also contains a set of queues such that one queue is associated with each input channel, allowing messages from any number of nodes to arrive and be buffered simultaneously. This organization supports multiple simultaneous broadcasts.

Messages exchanged between nodes contain a header field with information on the message type (data or synchronization), length, and destination address. Once the logic level signal is restored from the optical data, it is directed to the input channel interface, which consists of two parts: the optical interface, which includes physical signaling, address filtering, length monitoring, and type decoding and the processor interface, which includes a routing network and a queuing system. One queue is associated with each input channel, allowing messages from any number of processors to arrive and be buffered simultaneously, until the local processor is ready to remove them. Arbitration may be required only locally in a receiver array when multiple input queues contain messages.

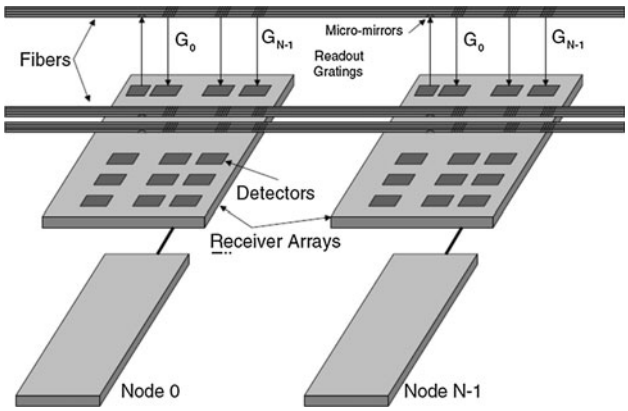

Fig. 1 Parallel receiver array

The SOME-Bus may appear to be equivalent to a mesh, but it has much more functionality. It avoids the latency of arbitration, switching setup, and informing the sending node that the connection is complete. The ability to support multiple simultaneous broadcasts is a unique feature of the SOME-Bus, which efficiently supports distributed barrier synchronization mechanisms and cache consistency protocols.

3 Message-passing protocol

A message-passing system typically combines the local memory and processor at each node of the interconnection network. There is no global memory so it is necessary to move data from one local memory to another by means of message-passing paradigm. This is typically done by a Send/Receive pair of commands, which must be written into the application software by a programmer. Thus, programmers must learn its complicated paradigm which involves data copying and dealing with consistency issues accurately. Simultaneous message processing and problem calculating are handled by the underlying operating system and protocols. Processes running on a given processor use mean named internal channels to exchange messages among themselves. Processes running on different processors use the external channels to exchange messages. Data exchanged among processors cannot be shared; it is rather copied (using send/receive messages). An important advantage of this form of data exchange is the elimination of the need for synchronization constructs, such as semaphores, which results in performance improvement. In addition, a message-passing scheme offers flexibility in accommodating a large number of processors [13].

The basic programming model used in message-passing architectures is based on the idea of matching a send

request on one processor with a receive request on another. In such scheme, send and receive are blocking; that is, send blocks until the corresponding receive is executed before data can be transferred. Message-passing communication protocol supports end-to-end packet acknowledgment. For every packet sent by a source node, there is a returned ACK's after the packet has reached the destination node. This allows source nodes to discover packet loss. Automatic retransmission of a packet is made if the ACK's is not received within a preset time interval. A message-passing programming style is the preferred style for performance on such model. Also, message passing without ACK's protocol can be defined as above neglecting the fact that the source is not in need learn whatever the sent packet has arrived or not (only broadcasts the packet). There are problems associated with message-passing systems. These include communication overhead and difficulty of programming [14–16].

4 Overview of methods

4.1 Multi-layer feed-forward artificial neural networks

The ANN employs the model structure of a neural network which is a powerful computational technique for modeling complex non-linear relationships particularly in situations where the explicit form of the relation between the variables involved is unknown [17, 18]. An MFANN consists of at least three layers—input, output, and hidden layer. The schematic diagram of a MFANN is shown in Fig. 2. Each neuron in a layer receives weighted inputs from a previous layer and transmits its output to neurons in the next layer. The summations of weighted input signals are calculated and this summation is transferred by a nonlinear activation function. The results of the network are compared with the actual observation results and the network error is trained until the error reaches an acceptable value [18].

In Fig. 2, X_i is the neuron input, W_{ij} and W_{kj} are the weights, M is the number of neurons in the hidden layer, and Y is the output value.

4.2 Generalized regression neural networks

The GRNN is a generalization of both radial basis function networks and probabilistic neural networks that can perform linear and nonlinear regression [19]. These feed-forward networks use basis function architectures which can approximate any arbitrary function between input and output vectors directly from training samples, and they can be used for multidimensional interpolation [20, 21]. The

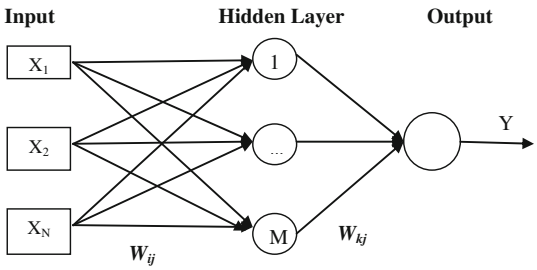

Fig. 2 A typical multi-layer feed-forward neural network architecture

main function of a GRNN is to estimate a linear or non-linear regression surface on independent variables (input vectors) U, given the dependent variables (desired output vectors) X. That is, the network computes the most probable value of an output, O_x, given only training vectors U. Specifically, the network computes the joint probability density function of U and X. The expected value of X given U is expressed as [19]:

$$E[X/U] = \frac{\int_{-\infty}^{\infty} Xf(U,X)\mathrm{d}x}{\int_{-\infty}^{\infty} f(U,X)\mathrm{d}x} \tag{1}$$

An important advantage of the GRNN is its simplicity and fast approximation procedure. Another attractive feature is that, unlike back propagation-based neural networks, GRNN does not converge to local minima [22]. The topology of a GRNN consists of four parts. First, there is an input layer that is fully connected to the pattern layer. Second, there is a pattern layer that has one unit for each pattern. It computes the pattern Gaussian function expressed by

$$h_i = \exp[-D_i^2/2\sigma^2]; \tag{2}$$

where

$$D_i^2 = (u - U_i)^T(u - U_i) \tag{3}$$

σ denotes the smoothing parameter, u is the input presented to the network, and U_i is each of the training vector. Third, there is a summation layer which has two units N and P. The first unit computes the weighted sum of the hidden layer outputs. The second unit has weights equal to "1," and therefore sums exponential terms (h_i) alone. Fourth, there is an output unit that divides N by P to provide the prediction result.

4.3 Support vector regression

4.3.1 Linear support vector regression

We are given the training data $(x_i, y_i), (i = 1, \ldots, l)$, where x is a d-dimensional input with $x \in \Re^d$ and the output is

$y \in \Re$. The linear regression model can be written as follows [23]:

$$f(x) = \langle \omega, x \rangle + b, \quad \omega, x \in \Re^d, \quad b \in \Re, \tag{4}$$

where $f(x)$ is an unknown target function and $\langle ., . \rangle$ denotes the dot product in $\Re^d$.

In order to measure the empirical risk [24], we should specify a loss function. The most common loss function is the ε-insensitive loss function proposed by Vapnik [25] and is defined by the following function:

$$L_\varepsilon(y) = \begin{cases} 0 & \text{for } |f(x) - y| \le \varepsilon \\ |f(x) - y| - \varepsilon & \text{otherwise} \end{cases} \tag{5}$$

The optimal parameters $\overline{\omega}$ and $\overline{b}$ in (4) are found by solving the primal optimization problem [26]:

$$\min \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{\ell}\left(\xi_i^- + \xi_i^+\right) \tag{6}$$

with constraints

$$\begin{aligned} &y_i - \langle \omega, x_i \rangle - b \le \varepsilon + \xi_i^+, \\ &\langle \omega, x_i \rangle + b - y_i \le \varepsilon + \xi_i^+, \\ &\xi_i^+, \xi_i^- \ge 0, \quad i = 1, \ldots, \ell \end{aligned} \tag{7}$$

where C is a pre-specified value which determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than the precision ε are tolerated. The slack variables ξ^- and ξ^+ represent the deviations from the constraints of the ε-tube.

Usually, the dual problem is solved. The corresponding dual optimization problem is defined as

$$\begin{aligned} \max_{\alpha,\alpha^*} &-\frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\left(\alpha_i^* - \alpha_i\right)\left(\alpha_j^* - \alpha_j\right)\langle x_i, x_j \rangle \\ &-\sum_{i=1}^{\ell} y_i\left(\alpha_i^* - \alpha_i\right) - \varepsilon\sum_{i=1}^{\ell}\left(\alpha_i^* + \alpha_i\right) \end{aligned} \tag{8}$$

with constraints

$$\begin{aligned} &0 \le \alpha_i, \alpha_i^* \le C, \quad i = 1, \ldots, \ell, \\ &\sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right) = 0 \end{aligned} \tag{9}$$

Solving the optimization problem defined by (8) and (9) gives the optimal Lagrange multipliers α and α^*, while $\overline{w}$ and $\overline{b}$ are given by

$$\begin{aligned} &\overline{\omega} = \sum_{i=1}^{\ell}\left(\alpha_i^* - \alpha_i\right)x_i, \\ &\overline{b} = -\tfrac{1}{2}\langle \overline{\omega}, (x_r + x_s) \rangle, \end{aligned} \tag{10}$$

where x_r and x_s are support vectors [26].

4.3.2 Nonlinear support vector regression

For nonlinear regression problems, a nonlinear mapping ϕ of the input space onto a higher dimension feature space can be used, and then linear regression can be performed in this space [27]. The nonlinear model is written as

$$f(x) = \langle \omega, \phi(x) \rangle + b, \quad \omega, x \in \Re^d, \quad b \in \Re, \tag{11}$$

where

$$\begin{aligned} &\overline{\omega} = \sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)\phi(x_i), \\ &\left\langle \omega, \overline{\phi}(x) \right\rangle = \sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)\langle \phi(x_i), \phi(x) \rangle \\ &= \sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)K(x_i, x), \\ &\overline{b} = -\frac{1}{2}\sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)\left(K(x_i, x_r) + K(x_i, x_s)\right) \end{aligned} \tag{12}$$

where x_r and x_s are support vectors. Note that we express dot products through a kernel function K that satisfies Mercer's conditions [25]. (11) can be written as follows if the term $\overline{b}$ is accommodated within the kernel function:

$$\sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)K(x_i, x) \tag{13}$$

Several kernel functions have appeared in the literature. The radial basis function (RBF) has received significant attention, most commonly with a Gaussian of the form:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\rho^2}\right). \tag{14}$$

where ρ is the width of the RBF kernel.

4.4 Multiple linear regression

MLR is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of MLR is to model the relationship between the explanatory and response variables. MLR models are often used in the prediction of network performance analysis, being represented by the relationship between network inputs and a set of predictor output variables in (15).

$$y_i = B_0 + B_1 x_{i1} + B_2 x_{i2} + \ldots + B_p x_{ip} + E_i \tag{15}$$

where $i = 1, 2, \ldots, n$; B_i is the residual, E_i is the difference between the value of the dependent variable predicted by the model and the dependent variable, and x is the independent parameter.

5 Simulation and dataset generation

OPNET Modeler [11] is used to simulate the SOME-Bus architecture employing the message-passing protocol with ACK's. Figure 3 shows the node model of the simulated architecture. Each node contains a processor station in which the incoming messages are stored and processed, and also a channel station in which the outgoing messages are stored before transferring them onto the network.

The underlying process model which controls queue modules' behavior is OPNET's built-in acb_fifo model which is shown in Fig. 4. The model has its own server and can concentrate multiple incoming packets streams into its single internal queuing resource. It also supports the First-in-First-out service ordering discipline and a way to control service times. The "init" state is used to initialize the process and setting the appropriate variables. If a packet arrives when the process is in "init" state, the process transitions to the "arrival" state, else it transitions to the "idle" state where it waits for packet arrival. The "arrival" state is used for receiving packets and starting service. In the "arrival" state, if the server is not busy then the process moves into the "svc_start" state, which in turn transitions to the "idle" state, where it waits either for packet arrival or service completion. While in the "idle" state, if the processing of a packet is completed, the process moves into the "svc_compl" state. While in the "svc_compl" state, if the queue is not empty, the process moves into the "svc_start" state.

The important parameters of the simulation are the number of nodes (selected as 16, 32, and 64), the number of the threads executed by each processor (ranging from 1 to 6), *T*/*R*, (ranging from 0.05 to 1), thread run time (exponentially distributed with a mean value of 100), and traffic pattern (uniform, hot-region, bit reverse, and perfect shuffle). In the uniform traffic, the destination node is selected using uniform distribution among the nodes. In the hot-region pattern, the destinations of the 25 % of the packets are chosen randomly within a small hot region consisting of 12.5 % of the nodes [12]. Bit permutations such as bit reverse and perfect shuffle are those in which each bit d_i of the *b-bit* destination address is a function of the one bit of the source address [8]. The dataset obtained as a result of the simulation contains four input and five output variables. The input variables of the prediction model include *T*/*R*, node number, thread number, and traffic pattern. The output variables of the prediction model include average CWT (i.e., the time interval between the instant when a packet is enqueued in the output channel until the instant when the packet goes under service), average CU (i.e., average fraction of time that the channel server is busy), average NRT (i.e., the time interval between the instant when a message is enqueued in the output channel until the instant when the corresponding acknowledge message arrives at the input queue), average PU (i.e., average fraction of time that threads are executing), and average IWT (i.e., the time interval between the instant when a message is enqueued in the input queue until the instant when the message gets service from the processor).

The dataset obtained as a result of the statistical simulation includes 792 samples. Table 1 gives the descriptive statistics of the dataset.

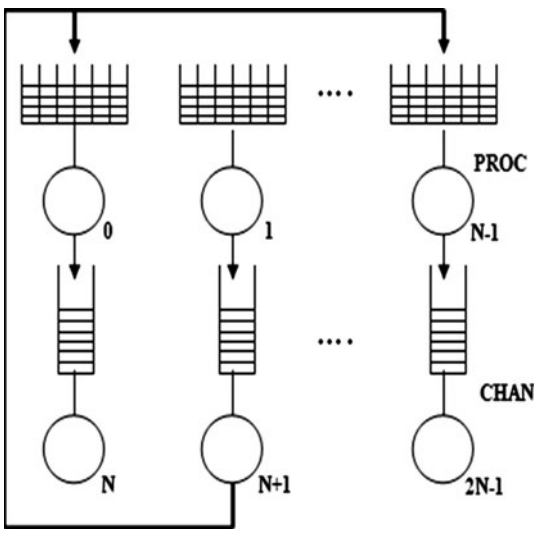

Fig. 3 A typical *N*-node SOME-Bus architecture by message-passing protocol

6 Results and discussion

The MFANN prediction model is shown in Fig. 5. As is seen in Fig. 5, the neural network structure contains two hidden layers. The first hidden layer has 9 neurons and the second hidden layer has 6 neurons. These numbers have been obtained by trial-and-error (i.e., after testing the

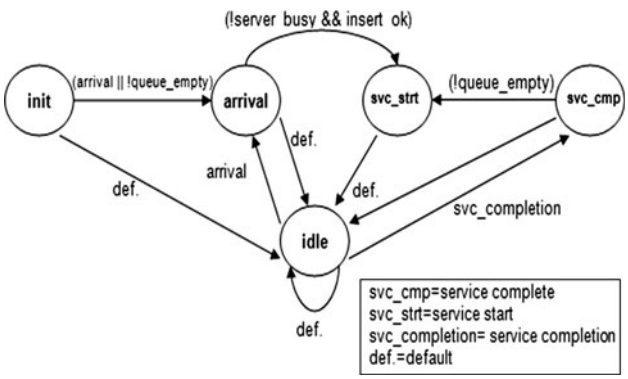

Fig. 4 A typical process model for the queues

Table 1 Descriptive statistics of the dataset

Statistics name	Performance measures				
	CWT	CU	NRT	PU	IWT
Mean	19.0801	0.2322	449.4143	0.4649	167.8480
Maximum	186.3973	0.8541	1,027.3580	0.9509	356.9148
Minimum	0.0031	0.0007	20.6056	0.0119	2.1585
SD	28.8380	0.2129	240.3182	0.2892	94.7545

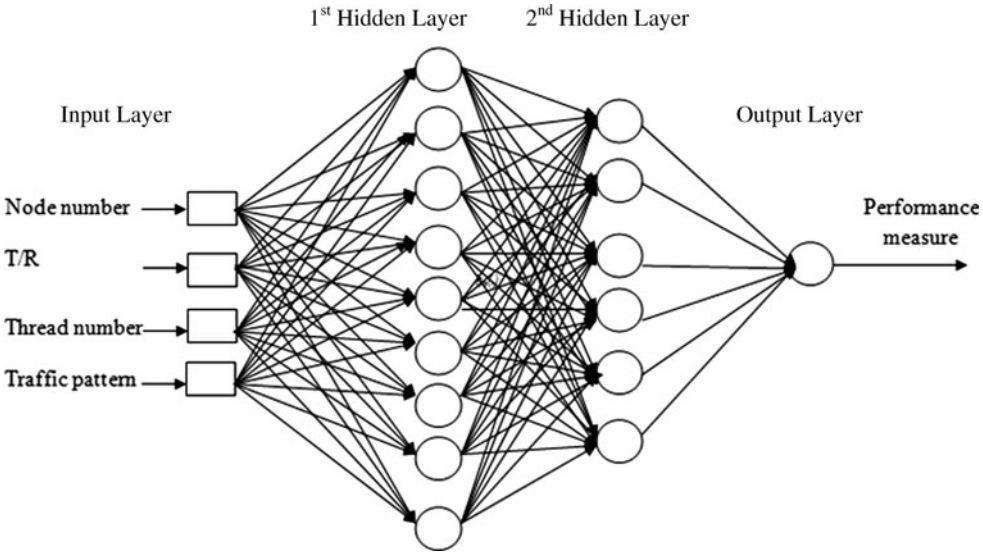

Fig. 5 MFANN model for prediction of the performance measures

neural network with several different configurations and observing that these numbers yield the lowest error rates for prediction). A tansigmoid activation function is used in the hidden layers. A pure-linear activation function is used in the output layer. The Levenberg–Marquardt (LM) algorithm is utilized for training the network. The other important parameters of the MFANN model are the number of epochs (selected as 500), the learning rate (selected as 0.02), and momentum (selected as 0.5).

The performance of the ANN prediction model is evaluated using R and SEE, the formulas of which are given in Eqs. (12) and (13), respectively

$$R = \sqrt{1 - \frac{\sum_{i=1}^{n} (Y - Y')^2}{\sum_{i=1}^{n} \left(Y - \bar{Y}\right)^2}} \tag{16}$$

$$\text{SEE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (Y - Y')^2} \tag{17}$$

where n is the number of data points used for testing, Y is the observed value, Y' is the predicted value and $\overline{Y}$ is the average of the observed values.

The results of the MFANN prediction model are compared with the ones obtained by GRNN, SVR, and MLR. Tables 2, 3, 4, 5, 6, 7, 8, and 9 show the performance of all prediction models in terms of R and SEE using different number of cross-validation folds. In Tables 2, 3, 4, 5, 6, 7, 8, and 9, RBF stands for radial basis function and L stands for linear.

Based on the results, the following points can be made:

1. For all performance measures, the MFANN-based prediction model performs better (i.e., higher R and lower SEE) than SVR-based, GRNN-based, and MLR-based prediction models.
2. SVR-RBF model shows the second best performance for prediction.
3. The SEE of the MFANN-based prediction model decreases as the number of folds in the test set increases from 10 to 80. However, it is observed that the SEE of the ANN-based model increases as the number of folds exceeds 80.
4. The MFANN-based model performs a perfect job in predicting CU and PU (i.e., the SEE is almost zero for both predictions). The prediction errors related to NRT and IWT are higher than the ones related to CWT. This is because of the high standard deviation of NRT and IWT in the dataset.
5. Although the MLR-based prediction model yields good performance for prediction of CU and PU, it does not show the same performance for prediction

Table 2 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L, and SVR-RBF models by means of tenfold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9956	0.9009	0.7719	0.7802	0.9926	2.5401	12.2491	17.7719	19.4299	3.5152
CU	0.9992	0.9888	0.8915	0.9911	0.9864	0.0088	0.0314	0.0948	0.0288	0.0573
NRT	0.9955	0.9927	0.7689	0.7712	0.9893	22.3406	28.0563	151.3559	153.5735	35.1874
PU	0.9993	0.9988	0.9568	0.9566	0.9883	0.0106	0.0143	0.0834	0.085	0.0566
IWT	0.9737	0.9554	0.8225	0.8251	0.9572	20.7994	27.6846	53.1487	53.6145	27.4746

Table 3 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 20-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9961	0.9107	0.7449	0.7794	0.9925	2.3610	11.3501	17.4882	20.9333	3.5404
CU	0.9992	0.9924	0.8869	0.9926	0.987	0.0081	0.0243	0.0941	0.0261	0.0571
NRT	0.9964	0.9934	0.76	0.7688	0.9899	19.9550	26.4159	151.9981	155.632	34.2684
PU	0.9994	0.9988	0.9548	0.9566	0.9883	0.0100	0.0134	0.0827	0.0851	0.0566
IWT	0.9757	0.958	0.8106	82.76	0.9609	19.5336	25.3917	53.4062	53.1712	26.234

Table 4 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 30-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9973	0.9191	0.7556	0.7781	0.9925	1.9756	10.4803	17.2198	20.9609	3.5278
CU	0.9994	0.9938	0.8827	0.9931	0.9866	0.0074	0.0223	0.094	0.0252	0.0572
NRT	0.9966	0.9933	0.7578	0.7685	0.9895	19.2222	25.9292	150.9108	155.6577	34.7925
PU	0.9994	0.9988	0.9493	0.9567	0.9884	0.0092	0.013	0.0831	0.0849	0.0566
IWT	0.9792	0.957	0.7959	0.8279	0.96	17.7565	24.4516	52.6166	53.1414	26.5457

Table 5 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 40-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9975	0.9264	0.7074	0.779	0.992	1.7644	9.7633	17.0384	20.9565	3.659
CU	0.9994	0.9927	0.8713	0.8977	0.9869	0.0068	0.0236	0.0939	0.0941	0.0572
NRT	0.9969	0.9923	0.7462	0.7692	0.9896	18.3303	26.5084	150.6976	155.4807	34.556
PU	0.9995	0.9988	0.9515	0.9566	0.9884	0.0090	0.0132	0.0812	0.0851	0.0565
IWT	0.9811	0.9531	0.7977	0.8279	0.9611	16.1882	23.3667	52.8934	53.1395	26.2092

of CWT, NRT, and IWT. This is because of the non-linear characteristics of CWT, NRT, and IWT.

6. Since there is no training phase in GRNN, the GRNN-based model produces results much faster than MFANN-based and SVR-based prediction models.
7. The MFANN-based prediction model yields the lowest SEE for prediction of PU, where the SEE changes from 0.0125 to 0.0143.
8. The MFANN-based prediction model yields the highest SEE for prediction of NRT, where the SEE changes from 14.2463 to 22.3406.

Table 6 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 50-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9973	0.9234	0.68	0.7788	0.9924	1.5529	9.3122	17.131	20.9541	3.5811
CU	0.9995	0.9922	0.8615	0.8976	0.9869	0.0064	0.0225	0.0931	0.0942	0.0573
N*RT*	0.9974	0.9923	0.7191	0.7688	0.9896	16.6480	26.3149	150.6653	155.5459	34.7366
PU	0.9995	0.9987	0.9429	0.9571	0.9885	0.0083	0.0128	0.0822	0.0852	0.0564
IWT	0.9836	0.9531	0.7929	0.8279	0.9609	14.5471	23.4087	51.7876	53.1319	26.249

Table 7 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 60-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9995	0.9264	0.6057	0.7791	0.9927	1.4062	9.4494	16.313	20.9487	3.4927
CU	0.9995	0.9934	0.8745	0.8976	0.987	0.0061	0.0213	0.0936	0.0941	0.0573
NRT	0.9974	0.9929	0.6925	0.7686	0.99	16.3740	25.3646	148.9668	155.6017	34.0167
PU	0.9995	0.9986	0.9406	0.9571	0.9885	0.0076	0.0127	0.0814	0.0852	0.0565
IWT	0.9851	0.9539	0.7769	0.8281	0.9616	13.6680	22.9766	52.0667	53.1063	26.0344

Table 8 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 70-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9981	0.9327	0.6057	0.7788	0.9921	1.3308	8.4841	16.313	20.9572	3.5788
CU	0.9995	0.9931	0.8214	0.8975	0.987	0.0057	0.0212	0.0924	0.0942	0.0571
NRT	0.9979	0.9918	0.7137	0.7693	0.99	14.8896	25.5296	149.8528	155.4193	33.9499
PU	0.9995	0.9982	0.9411	0.957	0.9885	0.0075	0.0125	0.0806	0.0853	0.0566
IWT	0.9893	0.9401	0.7298	0.8281	0.9619	12.5396	24.2335	51.7245	53.1109	25.9861

Table 9 *R* and SEE values of the MFANN, GRNN, MLR, SVR-L and SVR-RBF models by means of 80-fold cross-validation

Performance measures	*R*					SEE				
	ANN	GRNN	MLR	SVR-L	SVR-RBF	ANN	GRNN	MLR	SVR-L	SVR-RBF
CWT	0.9980	0.937	0.6198	0.7791	0.992	1.1782	8.1584	16.3505	20.9464	3.5854
CU	0.9996	0.9845	0.8578	0.9938	0.9868	0.0054	0.0213	0.091	0.0238	0.0573
NRT	0.9979	0.9915	0.6367	0.7678	0.9894	14.2463	24.5993	147.0421	154.4348	34.8452
PU	0.9994	0.9986	0.9103	0.894	0.9885	0.0072	0.0127	0.0811	0.0956	0.0565
IWT	0.9867	0.9357	0.757	0.8278	0.9618	11.9345	22.8651	51.0856	53.1457	25.9849

9. MLR and SVR-L models show similar performance for prediction.
10. The *R* values for prediction of CWT, CU, NRT, PU, and IWT are close to 1 for all folds.
11. The training times for MFANN-based and SVR-based models are given in Tables 10, 11, and 12. The training times for MFANN-based models are much lower than that of SVR-based models.
12. The training phase for SVR-RBF model consumes the longest time to make the predictions compared against the ones obtained by other models. This is because of the usage of the Gridsearch algorithm in the SVR-RBF model to compute the optimum values of the related parameters.
13. The execution times for the SVR-RBF and SVR-L prediction models change from 5 to 6 s, whereas the

Table 10 Training times for the MFANN models for different number of folds

Fold no.	Training times (s)				
	CWT	CU	NRT	PU	IWT
10	1.26	1.1700	1.1800	1.1500	1.2000
20	1.14	1.3200	1.1600	1.3500	1.1900
30	1.28	1.1700	1.1700	1.1700	1.2000
40	1.15	1.1800	1.3600	1.1600	1.2000
50	1.17	1.3400	1.2000	1.1500	1.2000
60	1.15	1.17	1.35	1.18	1.18
70	1.16	1.32	1.37	1.2	1.4
80	1.17	1.41	1.38	1.39	1.17

Table 11 Training times for the SVR-L models for different number of folds

Fold no.	Training times (s)				
	CWT	CU	NRT	PU	IWT
10	1,723.46	73.95	201.17	96.53	175.22
20	1,579.7	75.74	202.33	98.09	176.8
30	1,607.4	75.81	204.11	98.42	176.69
40	1,642.06	76.13	203.56	97.97	176.31
50	1,692.74	75.5	203.25	98.08	176.47
60	1,572.32	75.67	203.13	98.53	176.84
70	1,658.4	75.88	203.77	98.8	175.58
80	1,760.57	75.78	203.2	98.75	176.86

Table 12 Training times for the SVR-RBF models for different number of folds

Fold no.	Training times (s)				
	CWT	CU	NRT	PU	IWT
10	3,012.15	317.77	317.03	386.12	787.800
20	3,012.48	317.11	315.66	386.43	786.550
30	3,012.48	317.98	315.23	386.62	785.880
40	3,012.59	318.73	315.27	386.52	785.750
50	3,012.61	318.64	315.94	386.75	786.480
60	3,012.71	319.37	315.44	386.8	790.340
70	3,013.78	319.87	316.95	387.97	788.560
80	3,017.78	319.98	315.94	388.72	784.580

execution times for MFANN, GRNN, and MLR models are negligible (close to zero).

14. The global minima has been reached for all MFANN models.

7 Conclusion

This paper proposes to use MFANN's to predict the performance measures of a message-passing multiprocessor architecture. The basic idea is to collect a small number of performance measures by means of a statistical simulation and predict the performance of the system for a large set of input parameters based on these. OPNET Modeler is used to statistically simulate the message-passing SOME-Bus architecture. The obtained dataset contains five performance measures (i.e., NRT, CWT, PU, CU, and IWT) of the architecture. MFANN models with different number of folds have been developed to predict these performance measures. *R* and SEE values of the developed models have been calculated. The MFANN model gives the lowest SEE for the prediction of PU and the highest SEE for the prediction of the NRT. It is shown that MFANN models show better performance than GRNN-based, SVR-based, and MLR-based models for predicting the performance measures.

Future research can be performed in a number of areas. The first area would be expanding the number of input parameters in the dataset. The second area would be feature extraction on input variables. In this case, the critical attributes that best predict performance measures can be selected from a candidate set of attributes through feature selection algorithms combined with ANN's.

Acknowledgments We would like to thank the OPNET Technologies, Inc. for letting us use the OPNET Modeler under the University Program and to Cukurova University Scientific Research Projects Center for supporting this work (Project no: MMF2011D8). We would also like to thank Dr. Constantine Katsinis for letting us include the material about the SOME-Bus architecture in this paper.

References

1. Zhou X, Lu K, Wang X, Li X (2012) Exploiting parallelism in deterministic shared memory multiprocessing. J Parallel Distrib Comput 72:716–727
2. Culler D, Singh JP, Gupta A (2009) Parallel computer architecture: a hardware/software approach, 4th edn. Morgan Kaufmann, New York
3. Chow ALH, Golubchik L, Khuller S, Yaoc Y (2012) Performance tradeoffs in structured peer to peer streaming. J Parallel Distrib Comput 72:323–337
4. Chan F, Cao J, Sun Y (2003) High-level abstractions for message-passing parallel programming. Parallel Comput 29:1589–1621
5. Eeckhout L, Sampson J, Calder B (2005) Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In: Proceedings of the IEEE international symposium on workload characterization. Austin, TX, 6–8 October 2005, pp 2–12
6. Akay MF, Katsinis C (2008) Performance improvement of parallel programs on a broadcast-based distributed shared memory multiprocessor by simulation. Simul Model Pract Theory 16:338–352
7. Akay MF, Abasıkeleş I (2010) Predicting the performance measures of an optical distributed shared memory multiprocessor by using support vector regression. Expert Syst Appl 37: 6293–6630
8. Genbrugge D, Eeckhout L (2007) Statistical simulation of chip multiprocessors running multi-program workloads. In: Proceedings of the 25th international conference on computer design, Lake Tahoe, CA, 7–10 October 2007, pp 464–471
9. Zayid EIM, Akay MF (2012) Computing and estimating the performance measures of a message passing multiprocessor architecture by using artificial neural networks. In: Proceedings of the 2nd international conference on computation for science and technology, ICCST-2, Niğde, Turkey, 9–11 July 2012, pp 76–77
10. Akay MF, Zayid EIM (2011) Predicting the performance measures of a message passing multiprocessor architecture by using artificial neural networks. In: Proceedings of the 2nd international symposium on computing in science and engineering, ISCSE-2011, Kuşadası, Turkey, 1–4 June 2011, pp 53–58
11. OPNET Modeler Inc. (2012) OPNET University program. http://www.opnet.com/university_program
12. Katsinis C (2001) Performance analysis of the simultaneous optical multi-processor exchange bus. Parallel Comput 27: 1079–1115
13. Acacio ME, González J, García JM, Duato J (2002) The use of prediction for accelerating upgrade misses in cc-NUMA multiprocessors. In: Proceedings of the 11th international conference on parallel architectures and compilation techniques. Virginia, USA, p 155
14. Hesham E, Mostafa A (2005) Advanced computer architecture and parallel processing. Wiley, Hoboken
15. Thiele L, Wandeler E, Chakraborty S (2005) Performance analysis of multiprocessor DSPs: a stream-oriented component model. IEEE Signal Process Mag 22:38–46
16. Lemoff BE, Ali ME, Panatopoulos G, Flower GM, Madhavan B, Levi AFJ, Dolfi DW (2004) MAUI: enabling fiber-to-the-processor with parallel wavelength optical interconnects. J Lightwave Technol 22:2043–2054
17. Pratas F, Trancoso P, Sousa L, Stamatakis A, Shi G, Kindratenko V (2011) Fine-grain parallelism using multi-core, Cell/BE, and GPU systems. Parallel Comput 38:365–390
18. Chen M-S, Yen H-W (2011) Applications of machine learning approach on multi-queue message scheduling. Expert Syst Appl 38:3323–3335
19. Khashei M, Hamadani AZ, Bijari B (2012) A novel hybrid classification model of artificial neural networks and multiple linear regression models. Expert Syst Appl 39:2606–2620
20. Alpaydın E (2010) Introduction to machine learning, 2nd edn. MIT press, London
21. Firat M, Gungor M (2009) Generalized regression neural networks and feed forward neural networks for prediction of scour depth around bridge piers. Adv Eng Softw 40:731–737
22. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, San Francisco
23. Cherkassky V, Ma Y (2004) Comparison of loss functions for linear regression. In: Proceedings of the IEEE international joint conference on neural networks, pp 400–405
24. Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge
25. Vapnik VN (2000) The nature of statistical learning theory. Springer, New York
26. Gunn SR (1998) Support vector machines for classification and regression. Technical Report, Department of Electronics and Computer Science, University of Southampton, Southampton
27. Schölkopf B, Smola AJ (2002) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT Press, Cambridge