# VBMAR: Virtual Network Load Balanced Minimal Adaptive Routing

Xicheng Liu, Timothy J. Li, and Wen Gao
Motorola-NCIC Joint R&D Laboratory
Chinese Academy of Sciences
Beijing 100080, P.R. China
Email: xcliu@jdl.com.cn

## Abstract

*Fully adaptive routing with less virtual channels is of technical trend in MPP interconnection network design. However, imbalance of load offsets performances of many schemes. In this paper we give a fully adaptive load balanced routing algorithm called VBMAR. A new design tool called crossed turn model (CTM) is proposed to solve complex deadlock problems across virtual networks while keep the load balanced. Non-deterministic finite automaton (NDFA) is used to model the routing process. The resulting algorithm features routing uniformity and channel priority. Concurrent architecture and parallel priority channel searching techniques are used in VBMAR router design, which make it cost efficient. Four algorithms are compared to evaluate the performance of VBMAR. Simulations show that VBMAR can considerably increase network throughput and shorten network delay under both uniform and nonuniform traffic patterns. We argue that VBMAR is a very competitive routing scheme for future commercial MPP systems.*

## 1. Introduction

In multicomputer systems, the interconnection network implements message routing and flow control. Most of the current commercial and prototype systems still take deterministic routing (DR) schemes, such as X-Y algorithm [5]. However, an adaptive routing (AR) algorithm usually has much higher throughput and shorter delay. From the second generation of multicomputer wormhole flow control been a main flow control strategy instead of the store-and-forward [5]. It needs much less channel buffer and generates a network delay almost independent of network size. Virtual channels are extensively used in adaptive routing algorithms to solve deadlock problem. Because of the high cost of virtual channel implementation, a great deal of effort has been devoted to design algorithms using less virtual channels.

However, many proposed schemes are far from fully utilizing the potential of virtual channels in performance improvement. A key factor affecting the performance is load balance. [8] shows that imbalance of load can greatly reduces the performance. [1] supports this observation and indicates definitely that adaptive routing algorithm does not necessarily bring performance improvement unless other factors including load balance are properly treated. Most previous schemes with less virtual channels did not care this problem except a few efforts. [7] proposed an algorithm trying to distribute the load uniformly in different network regions. However, it did not carefully consider messages' travel across virtual networks, and would generally lead to a more complex router. Schemes in [6] and [4] allow message to travel across virtual networks on some certain conditions. However, the conditions are too strict, and network performance is thus limited.

In this paper we will design a new algorithm called VBMAR, which features load balance both among network regions and between virtual networks. We start from the turn model [3], a simple routing scheme employing least virtual channels. To get full adaptivity and increase network throughput, VBMAR employs two constituent virtual networks. A main concern in the design is how to solve the complex deadlock problems across virtual networks while keep the load balanced. We propose a graphical tool called crossed turn model to facilitate this work. The paper is arranged as follows. In section 2, we will describe the design methodology, crossed turn model, and state analysis of VBMAR in details. In section 3, a low cost, high speed adaptive router implementing the VBMAR algorithm will be presented. In section 4, we will evaluate the performance of VBMAR and identify the contribution of each of its constructs by comparing it with three other algorithms. In section 6 we will give conclusions.

## 2. Routing algorithm

### 2.1. Design methodology

In following description we name directions as in a geographical map, i.e., use E, W, N, and S to represent directions +X, -X, +Y, and –Y respectively. Turn from direction A to direction B is symbolized as A$\rfloor$B. According to the turn model, turns of messages in the network may form circles, which account for deadlock. Deadlock can be avoided by prohibiting at least one turn in a circle. Turn model is not fully adaptive as a routing model. In some region of the

network, it allows fully adaptive routing. In the rest region, it only allows deterministic routing. We call these regions adaptive region and non-adaptive region respectively. To get full adaptivity, we combine two complementary turn models, which are defined as follows.

**DEF1** Two turn models $TM_A$ and $TM_B$ are called *complementary* to each other if $TM_A$'s adaptive region is $TM_B$'s non-adaptive region and $TM_A$'s non-adaptive region is $TM_B$'s adaptive region. For example, west-first turn model $TM_{wf}$ and east-first turn model $TM_{ef}$ are complementary. They are actually the following turn sets.

$$TM_{wf} = \{W\_S, S\_E, E\_N, W\_N, N\_E, E\_S\}$$
$$TM_{ef} = \{W\_S, E\_N, N\_W, W\_N, E\_S, S\_W\}$$

Assume $(x_s, y_s)$ and $(x_d, y_d)$ are source node and destination node respectively. In region $x_d \geq x_s$ $TM_{wf}$ is fully adaptive while $TM_{ef}$ is non-adaptive, and the case is contrary in the rest region.

The full adaptivity can be realized in this way. First, divide a physical channel $c_i$ into two virtual channels $vc_i^{(1)}$ and $vc_i^{(2)}$. $C^{(1)} = \{vc_i^{(1)} \mid i = 0, 1, ..., N-1\}$ and $C^{(2)} = \{ vc_i^{(2)} \mid i = 0, 1, ..., N-1\}$ form two separate virtual networks. Here, N is the number of physical channels in the network. Second, assign two complementary turn models to $C^{(1)}$ and $C^{(2)}$, for example, $TM_{wf}$ to $C^{(1)}$ and $TM_{ef}$ to $C^{(2)}$. A message is routed in one of the virtual networks. The virtual network is so selected that the message can travel in it fully adaptively to its destination node. This can always be realized because total adaptive region of the two complementary turn models covers the whole network.

However, traffic load is not balanced in above scheme. To illustrate this problem, we first define a uniform format, "Σn", to represent all directions in the two virtual networks, which means direction Σ in virtual network $C^{(n)}$, n=1, 2. In minimal adaptive routing, message in $C^{(1)}$ always travels to the east part of the network. So direction W1 is never used. Similarly, message in $C^{(2)}$ always travels to the west part, and E2 is never used. Therefore, W1 and E2 always keep idle while all other directions are loaded with traffic. This is declined to decrease network throughput and cause hotspots, which limit network performance greatly.

To get load balanced, messages should be allowed to move across virtual networks. However, many more deadlock problems are aroused then. We found the cause is that the turn model of a virtual network allows turns prohibited by that of the other virtual network. Messages may take these turns in this virtual network and then enter the other. Thus the restriction on turns actually does not work. We can look upon any transfer across virtual networks as a turn too, and treat it equally with turns inside virtual networks. All turns should be taken into account as a whole in designing a deadlock free scheme.

We give a new graphical model to facilitate this work. The model is called a crossed turn model(CTM), as illustrated in figure 1. It is composed of two complementary turn models, one being surrounded by the other. Each constituent turn model is for a virtual network (The outer one is for $C^{(1)}$ ). The two turn models are not separate here, but combined as a whole by eight bridges (black bars in figure 1). Turns happen at bridges. A bridge equals four turns, i.e., turns inside $C^{(1)}$, $C^{(2)}$, from $C^{(1)}$ to $C^{(2)}$, and from $C^{(2)}$ to $C^{(1)}$. To avoid deadlock, what we have to do is just to find a bridge configuration without circles. So CTM provides a graphical tool to analyze deadlock when there exist virtual networks. The configuration without any restriction on turns across virtual networks is called original configuration. We can reach that the deadlock freedom probability for this configuration is 43.75%, and the deadlock probability is thus 56.25%. An idealized CTM should nullify this probability.

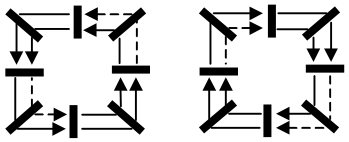

**Figure 1. A crossed turn model.**

## 2.2. Crossed turn model

Before referring to the algorithm details we will give some definitions.

**DEF2** The *home net* of a message is the virtual network in which the message can be routed fully adaptively.

**DEF3** The other virtual network except its home net is called the *auxiliary net* of a message.

**DEF4** A message is a *host message* of its home net.

**DEF5** A message is a *guest message* of its auxiliary net.

**DEF6** The *radial direction* in the west-first turn model is the west, and in the east-first turn model is the east. They are also *radial directions* of corresponding virtual networks.

**DEF7** A direction in a virtual network along which the network's host message can travel is called an *available direction* of the network.

**DEF8** All other directions except available directions in a virtual network are called *unavailable directions* of the network.

Minimal adaptive routing imposes more restrictions implicitly on turn model. Because it only allows hops that make less the distance to destination node, turns leaving minimal paths are actually prohibited. W_N and W_S in $TM_{wf}$ and E_N and E_S in $TM_{ef}$ fall into this category. We thus update the turn models as

$$TM_{wf} = \{S\_E, E\_N, N\_E, E\_S\}$$
$$TM_{ef} = \{W\_S, N\_W, W\_N, S\_W\}$$

These are complementary turn models for our CTM. The available directions of west-first virtual network are E, N and S, and those of east-first virtual network are W, N and S. Unavailable directions of them are of course W and E respectively. We can also work out that the deadlock probability for the original configuration based on them is 31.64%.

We set out from a basic version of algorithm called separate virtual net adaptive routing (SVAR). It views the network as two separate virtual networks. Messages are only routed in

their home nets in a fully adaptive manner. Then we focus on how to add turns across virtual networks to balance the load while keep the CTM deadlock free.

Our principle of balancing load is to make idle directions in SVAR share some load. At the same time, directions already having traffic would not be burdened further. In more details, directions W2 and E1 should hold some traffic now. Starting from this point we can deduce the whole CTM. It is depicted in figure 2. Solid arrows indicate turns allowable to cross virtual networks, and dash arrows indicate those prohibited. We give detailed rationale as follows.

1) Messages can use unavailable directions W1 and E2 in their auxiliary nets, so bridge $e_e$ and $e_w$ are allowed in figure 2. At the same time, available directions N1, N2, S1 and S2 can not be further burdened with guest messages. So bridges $e_n$ and $e_s$ are prohibited.

2) Messages along vertical directions in their home nets can also turn to unavailable directions W1 or E2 in their auxiliary nets to balance the load. So turns $e_{x1}$ = {N1⌋E2, S1⌋E2, N2⌋W1, S2⌋W1} are allowed. They can also turn back from their auxiliary nets to their home nets. So turns $e_{y2}$ = {W1⌋N2, W1⌋S2, E2⌋N1, E2⌋S1} are also allowed.

3) Because messages do not use vertical directions in their auxiliary nets, turns to these directions from radial directions in their home nets $e_{y1}$ = {E1⌋N2, W2⌋S1, E1⌋S2, W2⌋N1} are prohibited. Correspondingly, there are not turns from these directions to radial directions in their home nets. So $e_{x2}$ = {N1⌋W2, S1⌋W2, N2⌋E1, S2⌋E1} are also prohibited.

Checking the resulting CTM in figure 2, we can find no circle. So it can guarantee deadlock freedom.

We summarize above CTM as follows.

1) It is composed of two complementary turn models $TM_{wf}$ and $TM_{ef}$, each for a virtual network.

2) A message is routed in a minimal fully adaptive manner.

3) A message can turn from all available directions in its home net to the unavailable direction in its auxiliary net.

4) A message can turn from the unavailable direction in its auxiliary net to all available directions in its home net.

5) A message can only move along the unavailable direction in its auxiliary net.

This CTM utilize previously unavailable directions to balance the load while keep deadlock-free. We call the routing algorithm based on it a Virtual network load Balanced Minimal Adaptive Routing algorithm (VBMAR).

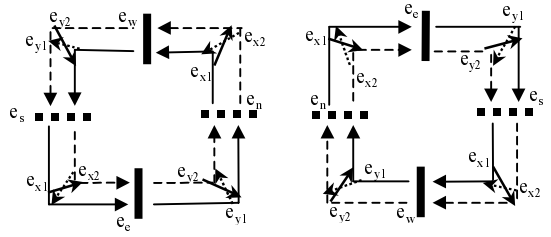


**Figure 2. Crossed turn model for VBMAR.**

## 2.3. State analysis

When a message is routed in the network, the routing channel *c* for next step is decided based on the message state Q and network input I. Q includes the message's current position relative to the destination node, D, current virtual network C, and current moving direction H. Network input I indicates availability of all possible channels for next step. Using these information, the routing algorithm determines the next channel *c* by further referring to the CTM. When a hop is finished, the message transfers to a new state. So the routing process can be modeled as a non-deterministic finite automaton (NDFA), which is expressed as the following quintuple:

$$G = ( Q, I, R, q_0, q_e )$$

Meanings of symbols in this formula are as follows.

Q: message state set, Q = D×C×H

I: network input set

R: state translation function

$q_0$: initial state of the message

$q_e$: ending state of the message

The translation function R: Q×I→Q is just the routing algorithm embodying the CTM in figure 2.

There are nine possible positions of a message relative to its destination node, as shown in figure 3(b) by small circles. The pentacle indicates the destination. So D has nine elements. C has two elements, $C^{(1)}$ and $C^{(2)}$. H has eight elements, i.e., H={E1, E2, N1, N2, W1, W2, S1, S2}. Therefore, there are totally 9×2×8=144 possible message states in set Q = D×C×H. This makes the NDFA very complex. Fortunately, VBMAR has a salient feature that it is independent of C and H. From above CTM we can see that whichever virtual network and direction a message is in, possible channels for next routing step are the same. In state D1, for example, the output channel set $C_{next}$ for messages from different directions E1, E2 and N1 are all $C_{next}$ = {E1, E2, N1}. So we can decide the next channel only based on D. We call this property uniformity of the routing algorithm. It can greatly simplify the NDFA, decreasing the number of state into only nine. The resulting NDFA is shown in figure 3(a). Each state corresponds to an element of D. Starting from an initial state $q_0$, a message can transfer to neighboring states until the ending state $q_e$ is reached. In fact, $q_e$ is the state D5. Because messages are routed along minimal paths, not all states have to be traveled.

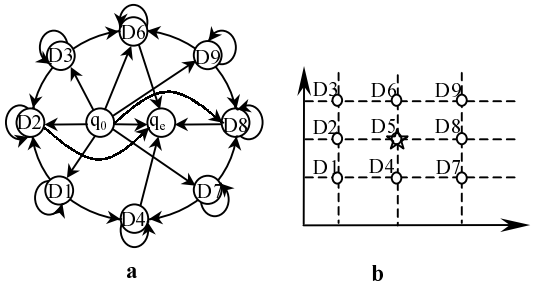


**Figure 3. NDFA of VBMAR (a) and message's relative position associated with each NDFA state (b).**

```
Name: VBMAR
Input: (d_x , d_y), home_net
Output: (vn, vc)

D = Match_state(d_x , d_y)
i = home_net
if D = D1
  if output vc=E1 is available
    return ( C^(1) ,vc)
  else if output vc=E2 is available
       return ( C^(2) ,vc)
     else if output vc=N1 is available
          return ( C^(1) ,vc)
        else
          return (NIL, NIL)
if D = D2
  if output vc=E1 is available
    return ( C^(1) ,vc)
  else if output vc=E2 is available
       return ( C^(2) ,vc)
     else
       return (NIL, NIL)
if D = D3
  if output vc=E1 is available
    return ( C^(1) ,vc)
  else if output vc=E2 is available
       return ( C^(2) ,vc)
     else if output vc=S1 is available
          return ( C^(1) ,vc)
        else
          return (NIL, NIL)
if D = D4
  if output vc=Ni is available
    return ( C^(i) ,vc)
  else
    return (NIL, NIL)
if D = D5
  if output vc=DEL is available
    return (DEL, DEL)
  else
    return (NIL, NIL)
if D = D6
  if output vc=Si is available
    return ( C^(i) ,vc)
  else
    return (NIL, NIL)
if D = D7
  if output vc=W2 is available
    return ( C^(2) ,vc)
  else if output vc=W1 is available
       return ( C^(1) ,vc)
     else if output vc=N2 is available
          return ( C^(2) ,vc)
        else
          return (NIL, NIL)
if D = D8
  if output vc=W2 is available
    return ( C^(2) ,vc)
  else if output vc=W1 is available
       return ( C^(1) ,vc)
     else
       return (NIL, NIL)
if D = D9
  if output vc=W2 is available
    return ( C^(2) ,vc)
  else if output vc=W1 is available
       return ( C^(1) ,vc)
     else if output vc=S2 is available
          return ( C^(2) ,vc)
        else
          return (NIL, NIL)
else
  return (NIL, NIL)


Meanings of Symbols:
( d_x, d_y): displacement of message's destination to
        current position
home_net: no. of message's home net
vn: virtual network selected for next hop
vc: virtual channel selected for next hop
Match_state(d_x , d_y): primitive to match message's
        current state with (d_x , d_y)
DEL: virtual channel to local processor
```

**Figure 4. VBMAR algorithm.**

### 2.4. Algorithm

Pseudo code of VBMAR is listed in figure 4. We can see from the algorithm that in each step output channels are not treated equally. Horizontal directions are searched first and vertical directions second. This priority favors the previously unavailable directions W1 and E2, and helps to balance the load and improve the channel utilization. The priority channel searching can be parallel executed in the router we designed, which will be described in next section.

## 3. Implementation Features

VBMAR router adopts a concurrent architecture. Any physical channel except the one connecting to local processor is shared by two virtual channels. Each virtual channel is implemented as a data bus inside the router. So two data buses serve one physical channel. A control pipeline is set for each physical channel, which controls the moving of a message through the router. Thus multiple messages can be routed simultaneously. A control pipeline includes three stages, input control logic, routing control logic, and output control logic. Each stage drains a clock cycle. Message moves in the router stage by stage under control. A data bus is actually a data pipeline. A message spends 3 clock cycles to go through a router when no blocking exists. This concurrent architecture can improve the network throughput greatly than the commonly used single control logic architecture.

We use a routing table to implement the routing control logic. This is different from most schemes that hide the routing logic deeply behind the customed circuit. It is both highly efficient to implement and easy to understand and design. As shown in table 1, each line of the routing table is a 9-bit routing vector corresponding to a NDFA state. Each bit of the vector is for an output direction, marking whether it is allowed for routing ('1') or not ('0'). When a message is read in, it is first decoded. Then the header flit is matched to generate a state number, which acts as an index to look up the routing table. The corresponding routing vector is thus activated. There is another vector called output mask, which records availability of each output port at the time. The routing vector together with the output mask determines the output port for next hop.

**Table 1. Routing table of VBMAR.**

| | E1 | W1 | N1 | S1 | E2 | W2 | N2 | S2 | DEL |
|---|---|---|---|---|---|---|---|---|---|
| $V_{D1}$ | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| $V_{D2}$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $V_{D3}$ | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $V_{D4}$ | 0 | 0 | * | 0 | 0 | 0 | * | 0 | 0 |
| $V_{D5}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $V_{D6}$ | 0 | 0 | 0 | * | 0 | 0 | 0 | * | 0 |
| $V_{D7}$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| $V_{D8}$ | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $V_{D9}$ | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

*: Its value is message dependent, being 0 or 1.

Several output ports may be available at the same time, and only the one with the highest priority should be selected. VBMAR router uses a parallel priority searching mechanism

to implement this efficiently. A sample logic is shown in figure 5. It is for state D1. In the logic, routing vector $V_{D1}$ ANDs output mask $V_M$ to pick all available ports out. At the same time, resulting bit for ports with higher priorities suppress results for those with lower priorities. Thus the resulting vector $V_O$ has either a single '1' bit within it, which indicates the fittest output port, or no '1' bit at all, which means no port is available. The overall logic expressions are as follows.

$$E1_{V_O} = E1_{V_{D1}} \cdot E1_{V_M}$$
$$E2_{V_O} = E1_{V_O} \cdot E2_{V_{D1}} \cdot E2_{V_M}$$
$$N1_{V_O} = E2_{V_O} \cdot N1_{V_{D1}} \cdot$$

Each item represents a bit in the vector its footmark indicates. Resulting vector $V_O$ serves as the address word to control the crossbar to set up connections for data forwarding.

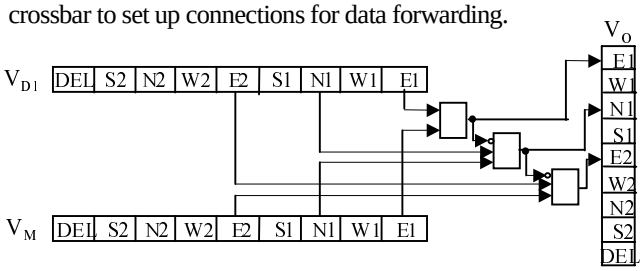


**Figure 5. Logic for parallel priority channel searching.**

There are four '*' in the routing table whose values are different from message to message. They correspond to state D4 and D6. A message in these states has finished all horizontal hops and has to walk along vertical directions in successive steps. It can not go to its auxiliary net any longer. So value of '*' is '1' for the direction in the message's home net, and '0' for that in its auxiliary net. The home net of a message can be judged by the routing control logic. However, this will increase the complexity because such a logic should be set for every routing pipeline. We simplify it in an efficient way. A message's home net is determined at its source node, and represented with a separate bit in its header flit.

In summary, the control logic for VBMAR is simple. It can realize high performance at a low cost. We can demonstrate its peak performance using A. A. Chien's model [2], which computes delay of every logic module of a wormhole router. It is assumed that the 0.8μm CMOS is used. Based on the model, delays of input, routing, and output control logic of VBMAR router are 3.68, 7.2, and 5.14 nanoseconds respectively. Clock cycle of the router should be larger than the delay of its slowest stage. So we set the clock cycle T = 10ns > MAX(3.68, 7.2, 5.14 ) = 7.2 ns. Thus the clock frequency is 100MHz. Because a physical channel includes 18 data wires, the bandwidth of a single channel is more than 200MB/s or 3.2Gb/s.

## 4. Performance Evaluation

In this section we will evaluate the performance of VBMAR algorithm under different traffic patterns. VBMAR adopts three mechanisms, virtual channel, adaptivity, and load balance. These are constructs of the algorithm. We will not only evaluate the overall performance of VBMAR, but also identify how much each construct contributes. Four algorithms will be compared for this purpose. They are X-Y, VDR, SVAR and VBMAR algorithms. Virtual net deterministic routing algorithm (VDR) is the same with SVAR except that it takes deterministic routing in each separate virtual network. The four algorithms add constructs in an incremental way, as listed in table 2. So comparing them can disclose the importance of each construct.

**Table 2. Features of four routing algorithms.**

| Algorithm | Virtual channel? | Adaptive? | Balanced? |
|---|---|---|---|
| X-Y | No | No | No |
| VDR | Yes | No | No |
| SVAR | Yes | Yes | No |
| VBMAR | Yes | Yes | Yes |

Uniform and nonuniform traffic patterns are used for performance evaluation. These traffic patterns can model a general class of message sources in MPP systems. They are used extensively in performance evaluation of interconnection networks. In uniform pattern, each node sends messages to all other nodes with the same probability. In nonuniform pattern, each node sends messages much more possibly to some certain nodes than to others. Thus these hotspots absorb most of communications in the network. In following simulations we will choose the central node of the network as the hotspot.

Simulations are based on a 16×16 mesh. The packet length is 20 flits. Buffer size for each virtual channel is 1 flit. Virtual channels sharing the same physical channel are arbitrated in a round-robin manner. The free or blocked virtual channel will not be allocated a time slot. Transmission delay of a physical channel between two neighboring routers is 1 clock cycle, and delay of a router without blocking is 3 cycles.

We use two metrics to measure network performance, network delay and network throughput. Network delay is the average delay of all messages through the network. Network throughput is the average number of messages delivered in a unit time. We normalize it as the proportion of this number to the number of total messages injected into the network in a unit time. We represent these metrics as functions of the normalized load of the network. The normalized load is the proportion of the bandwidth utilized to total bandwidth of the network [1].

Simulation results are shown in figure 6. They show clearly advantage of VBMAR to other algorithms, and contribution of each construct to network performance. We summarize them as follows.

1) VBMAR outperforms much all other algorithms under all traffic patterns. The order in terms of performance level from the highest to the lowest is VBMAR, SVAR, VDR, and X-Y.

2) VDR shortens the network delay considerably compared with X-Y. SVAR decreases it further. This suggests that virtual channel is a key factor affecting network delay, and adaptivity can enhance virtual channels' utilities very much. Nevertheless, they are not as important to network throughput improvement. We see X-Y, VDR and SVAR bear similar

network throughputs.

3) While SVAR improves little in network throughput compared with VDR and X-Y, VBMAR produces a much higher result. The only difference between them is that VBMAR gets load balanced while SVAR not. This indicates clearly how important the load balance is to network throughput.

5) By combing virtual channels, adaptivity and load balance, VBMAR exceeds the dominant commercial routing algorithm X-Y very much. It results in a very low network delay when the normalized load is over 0.2. By contrast, X-Y algorithm drives the network into saturation in this scope. Critical load to saturation for VBMAR is as high as 0.45, which is twice higher than that for X-Y.

## 5. Conclusions

We have designed a high performance fully adaptive load balanced routing algorithm adopting two virtual networks, VBMAR. Crossed turn model has been proposed to solve complex deadlock problems across virtual networks while keep load balanced. Concurrent architecture and parallel priority searching techniques are used in VBMAR router design. These together with the uniformity of the algorithm make the router cost efficient. Simulations showed that load balance is of great importance in performance improvement. By combing virtual channels, adaptivity, and load balance, VBMAR outperforms other algorithms much. It is a very competitive scheme to substitute the X-Y router in commercial MPP systems. In future work we will refine the algorithm and demonstrate its robustness to different network topologies and network sizes.

## References


[1] R. V. Boppana and S. Chalasani. A Comparison of Adaptive Wormhole Routing Algorithms, *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 351-360, 1993.

[2] A. A. Chien. A Cost and Speed Model for k-ary n-cube Wormhole Routers, *Proceedings of Hot Internnects'93*, August 1993.

[3] C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing, *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 278-287, May 1992.

[4] C. J. Glass and L. M. Ni. Maximally Fully Adaptive Routing in 2D Meshes. *Proceedings of International Conference on Parallel Processing*, pp. 101-104, 1992.

[5] Kai Hwang. *Advanced Computer Architecture - Parallelism, Scalability and Programmability*, New York: McGraw Hill, 1993.

[6] L. Schwiebert and D. N. Jayasimha. Optimal Fully Adaptive Wormhole Routing for Meshes, *Proceedings of Supercomputing'93*, pp. 782-791, 1993.

[7] J. H. Upadhyay, V. Varavithya, and P. Mohapatra. Efficient and Balanced Adaptive Routing in Two-Dimensional Meshes, *HiPC'96: Proceedings of High Performance Computing*, pp. 112-121, 1996.

[8] T. D. Nguyen and L. Snyder. Performance Analysis of a Minimal Adaptive Router, *Proceedings of the 1994 Parallel Computer Routing and Communication Workshop*, pp. 31-44, 1994.


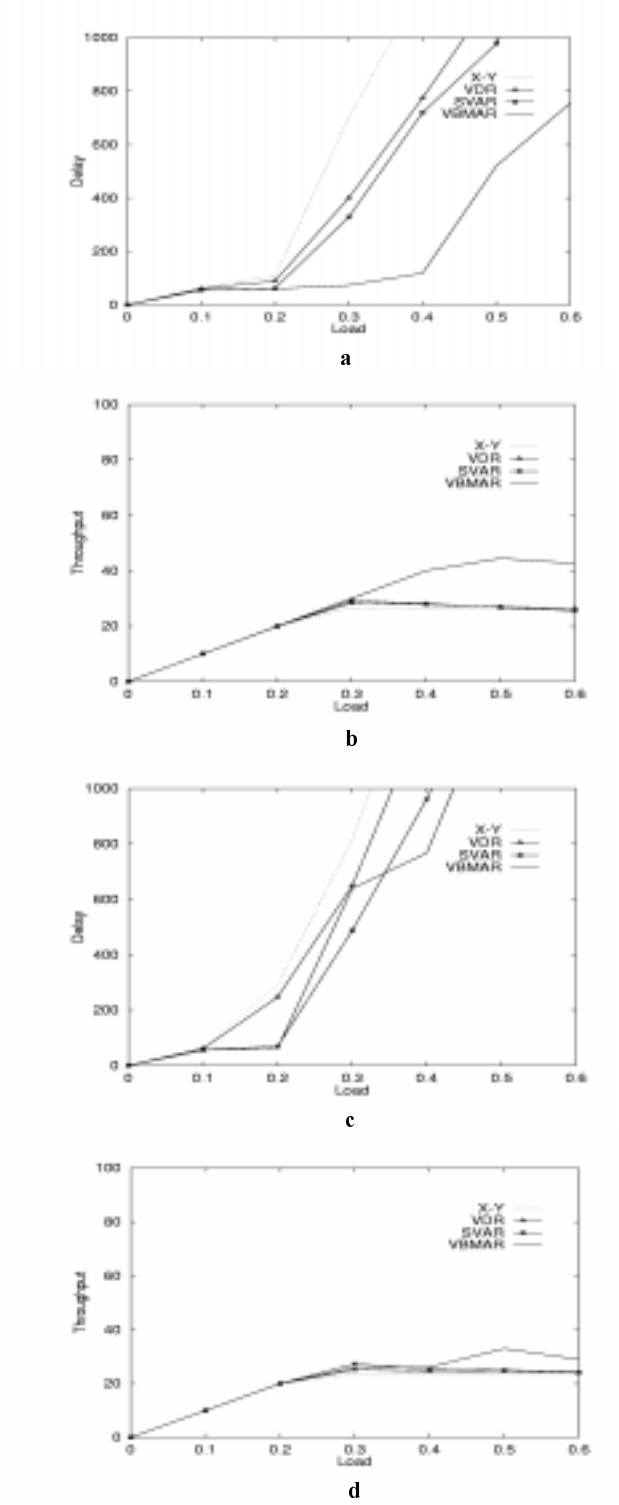


**Figure 6. Simulation results for four routing algorithms: network delays under uniform traffic (a), network throughputs under uniform traffic (b), network delays under nonuniform traffic (c), and network throughputs under nonuniform traffic (d).**