An Efficient Deadlock-Free Tree-Based Routing Algorithm for Irregular Wormhole-Routed Networks Based on the Turn Model

Yau-Ming Sun, Chih-Hsueh Yang, Yeh-Ching Chung[1], and Tai-Yi Huang
Department of Computer Science,
National Tsing Hua University,
HsingChu, Taiwan 300 R.O.C
{sym87u, dickyang}@sslab.cs.nthu.edu.tw, {ychung, tyhuang}@cs.nthu.edu.tw

Abstract

In this paper, we proposed an efficient deadlock-Free tree-based routing algorithm, the DOWN/UP routing, for irregular wormhole-routed networks based on the turn model. In a tree-based routing algorithm, hot spots around the root of a spanning tree and the uneven traffic distribution are the two main facts degrade the performance of the routing algorithm. To solve the hot spot and the uneven traffic distribution problems, in the DOWN/UP routing, it tries to push the traffic downward to the leaves of a spanning tree as much as possible and remove prohibited turn pairs with opposite directions in each node, respectively. To evaluate the performance of DOWN/UP routing, the simulation is conducted. We have implemented the DOWN/UP routing along with the L-turn routing on the IRFlexSim0.5 simulator. Irregular networks that contain 128 switches with 4-port and 8-port configurations are simulated. The simulation results show that the proposed routing algorithm outperforms the L-turn routing for all test samples in terms of the degree of hot spots, the traffic load distribution, and throughput.

1. Introduction

The *Up*/Down* routing* was the first deadlock-free tree-based routing algorithm for irregular networks. It is simple and easy to be implemented. However, it does not perform well. The reasons are three-fold. First, the so called *hot spots* [5] will occur around the root of a spanning tree. Second, the average length of routing paths is long compared to other tree-based routing algorithms. Third, there may exist two prohibited turns whose directions are opposite to each other on a node. These opposite prohibited turn pairs make the traffic distribution uneven [4].

To overcome the drawbacks of the *up*/down* routing*, the *L-turn routing* was proposed in [4] based on the 2D turn model [3]. In the *L-turn routing*, the routing is based on the *L-R tree*. By carefully setting up the prohibited turns for each node, one can obtain a more even distribution of traffic load and shorter routing paths compared to the *up*/down* routing* [4]. However, in the *L-turn routing*, the *tree links* and the *cross links* are considered as the same type of links. They have the same definition on directions. It is possible that the hot spots will still occur around the root under some *L-R trees* and makes traffic load unbalancing.

In this paper, we propose an efficient deadlock-free tree-based routing algorithm, the *DOWN/UP routing*, for irregular wormhole-routed networks based on the turn model. In the *DOWN/UP routing*, the *communication graph* based on the *coordinated tree* of a given topology is used to derive the prohibit turns. There are many ways to construct the coordinated tree of a given topology. Different coordinated trees lead to different performance of a routing algorithm. In this paper, we proposed a better way to construct the coordinated tree such that a better throughput can be obtained when a routing algorithm is performed.

In a communication graph, the tree links and cross links are considered as different links. They have different definitions on directions. This allows us to make the prohibited turn selection more precisely, that is, we can select the prohibited turns between the same or different type of links. Therefore, the traffic flow control by selecting prohibited turns in a communication graph is easier than that in the *L-R tree* used by the *L-turn routing*.

Based on the maximal direction graph, we can construct a maximal *acyclic direction dependency graph* by performing prohibited turn selection process. In the prohibited turn selection process, we careful select the set of prohibited turns such that the traffic can be pushed downward to leaves of a spanning tree as much as possible and a more even distribution of traffic load can be achieved. From the maximal acyclic direction dependency graph, the *DOWN/UP routing* can be derived by applying the same prohibited turns to each node in the communication graph and releasing unnecessary prohibited turns for each node. The *DOWN/UP routing*

[1] The corresponding author.

can be directly applied to arbitrary topology with (or without) any virtual channel of wormhole-routed networks.

To evaluate the proposed *DOWN/UP routing*, we compare its performance with that of *L-turn routing*. The hot spots, the traffic load distribution, and throughput are the three key factors for the evaluation. We implemented these two routing algorithms on simulated irregular networks that contain 128 switches with 4-port and 8-port configurations. The simulation results show that the *DOWN/UP routing* has less hot spots, better traffic load distribution, and higher throughput than the *L-turn routing*.

The rest of the paper is organized as follows. In Section 2, a brief survey of related work will be presented. In Section 3, we will introduce notations and terminology used in this paper. Section 4 presents the proposed deadlock-free routing algorithm in detail. The simulation environment and results will be presented in Section 5.

2. Related Work

Many deadlock-free routing algorithms for irregular network topologies have been proposed in the literature. In [7], the *up*/down* routing* used in DEC AN1 system was proposed for arbitrary network topologies. This adaptive routing algorithm assigns each network channel an *up* or a *down* direction based on a spanning tree. From the direction definitions, each packets must go through zero or more than one *up* direction channels followed by zero or more than one *down* direction channels to guarantee connectivity and deadlock-free. In [8], the *up*/down* routing* was extended to consider virtual channels or additional physical channels to achieve high-performance routing. In [6], an improved *up*/down* routing* based on depth first search (DFS) spanning tree was proposed. The proposed routing is deterministic source routing that has lower latency and better throughput than the original *up*/down* routing* algorithm.

Ni and Glass [1] proposed the *turn model* to design partially adaptive wormhole routing algorithms without any additional physical links and virtual channels. The idea of this model is to prohibit the minimum number of turns to break all of the cycles so that the routing is deadlock-free. Based on the model, the authors also proposed three partially adaptive routing algorithms, *west-first*, *north-first*, and *negative-first*, for two-dimensional meshes.

In [3] [4], Funahashi *et al*. proposed the *2D turn model* by introducing two-dimensional directions into a spanning tree to solve the traffic unbalancing problem occurred in the *up*/down* routing*. Two adaptive routing algorithms, *left/right routing* and *L-turn routing*, were proposed. The simulation results in [4] have shown that the *L-turn routing* achieves better performance than the *up*/down* routing*.

3. Preliminaries

<u>Definition 1</u>: A network with arbitrary switch-based interconnection can be represented as a graph $G = (V, E)$, where V is the set of switches and E is the set of bidirectional links between switches. For each link $e = (v_1, v_2)$ in E, it consists of two communication channels $<v_1, v_2>$ and $<v_2, v_1>$ in which v_1 can send message to v_2 through $<v_1, v_2>$ and v_2 can send message to v_1 through $<v_2, v_1>$. For each channel $<v_1, v_2>$, v_1 and v_2 are called *start* and *sink* nodes of the channel, respectively. $<v_1, v_2>$ is called the *output* channel and *input* channel of v_1 and v_2, respectively. G is called the network topology or the topology of the network.

<u>Definition 2</u> (*coordinated tree*): Given $G = (V, E)$, a *coordinated tree* is a breath first search (BFS) spanning tree of G. For each node v in a coordinated tree, node v is associated with a two dimensional coordinate $v(x, y)$. We use $X(v)$ and $Y(v)$ to denote the x and y coordinates of node v, respectively, that is, $X(v) = x$ and $Y(v) = y$. $X(v)$ is defined as the order of preorder traversal of the coordinated tree starting from the root to node A and $Y(v)$ is defined as the level of node v in the coordinated tree.

Due to two or more children nodes can be selected as the next preorder traversal node, several coordinated trees can be built from the same network topology.

<u>Definition 3</u>: Given $G = (V, E)$ and a coordinated tree $CT = (V, E')$ of G, E' and $E - E'$ are the sets of *tree links* and *cross links* of G with respect to CT, respectively.

<u>Definition 4</u>: Given $G = (V, E)$ and a coordinated tree $CT = (V, E')$ of G, for each link $e = (v_1, v_2) \in E$, we define

(1) v_2 is the *left-up* node of v_1 if $X(v_2) < X(v_1)$ and $Y(v_2) < Y(v_1)$,

(2) v_2 is the *left* node of v_1 if $X(v_2) < X(v_1)$ and $Y(v_2) = Y(v_1)$,

(3) v_2 is the *left-down* node of v_1 if $X(v_2) < X(v_1)$ and $Y(v_2) > Y(v_1)$,

(4) v_2 is the *right-up* node of v_1 if $X(v_2) > X(v_1)$ and $Y(v_2) < Y(v_1)$,

(5) v_2 is the *right* node of v_1 if $X(v_2) > X(v_1)$ and $Y(v_2) = Y(v_1)$, and

(6) v_2 is the *right-down* node of v_1 if $X(v_2) > X(v_1)$ and $Y(v_2) > Y(v_1)$.

<u>Definition 5</u> (*Communication graph* (*CG*)): Given $G = (V, E)$ and a coordinated tree $CT = (V, E')$ of G, the *communication graph* $CG = (V, \vec{E})$ with respect to G and CT is a directed graph, where $\vec{E}$ is the set of all communication channels of E. For each channel $\vec{e} = <v_1, v_2> \in \vec{E}$, if $\vec{e}$ is a channel of a tree link, the direction of

$\vec{e}$, denoted as $d(\vec{e})$, is defined as *LU_TREE* and *RD_TREE* if v_2 is a *left-up* node of v_1 and v_2 is a *right-down* node of v_1, respectively. If $\vec{e}$ is a channel of a cross link, the direction of $\vec{e}$ is defined as *LU_CROSS*, *LD_CROSS*, *RU_CROSS*, *RD_CROSS*, *R_CROSS*, and *L_CROSS* if v_2 is a *left-up* node of v_1, v_2 is a *left-down* node of v_1, v_2 is a *right-up* node of v_1, v_2 is a *right-down* node of v_1, v_2 is a *right* node of v_1, and v_2 is a *left* node of v_1, respectively.

Definition 6 (*Turn*): Given a communication graph $CG = (V, \vec{E})$, the directions of $\vec{e}_1$ and $\vec{e}_2$ form a *turn* for v_2 if $\vec{e}_1 = <v_1, v_2>$ and $\vec{e}_2 = <v_2, v_3>$. We use $T_{d(\vec{e}_1),d(\vec{e}_2)}$ to denote the turn formed by the directions of $\vec{e}_1$ and $\vec{e}_2$.

Definition 7 (*Turn cycle*): A *turn cycle TC* = $(T_{d(\vec{e}_1),d(\vec{e}_2)}, T_{d(\vec{e}_2),d(\vec{e}_3)}, \ldots, T_{d(\vec{e}_k),d(\vec{e}_{k+1})})$ is a sequence of turns in which the sink node of the first channel is also the sink node of the last channel in the turn sequence, that is, the start node of $\vec{e}_2$ is the sink node of $\vec{e}_{k+1}$.

Definition 8 (*Direction graph* (*DG*)): The *direction graph* $DG = (D, \vec{T})$ with respect to a communication graph $CG = (V, \vec{E})$ is a complete directed graph, where D is the set of directions defined in *CG* and $\vec{T} = \{T_{d_1,d_2} \mid$ for all $d_1, d_2 \in D$ and $d_1 \neq d_2\}$ is the set of possible turns that can be defined in *CG*. A *DG* is called the *complete direction graph* (*CDG*) if *D* = {*LU_TREE*, *RD_TREE*, *LD_CROSS*, *RU_CROSS*, *R_CROSS*, *L_CROSS*, *LU_CROSS*, *RD_CROSS*}.

Definition 9 (*Direction dependency graph* (*DDG*)): Given a *DG*, any subset of *DG* is called the *direction dependency graph* (*DDG*) of *DG*.

Definition 10 (*Acyclic direction dependency graph* (*ADDG*)): Given a *CG*, the *DG* of *CG*, and a *DDG* of *DG*, for each node *v* in *CG*, if the edges of *DDG* are the only available turns allowed at *v* and no turn cycle can be formed in *CG*, then the *DDG* is called *acyclic DDG*.

Definition 11: Given a *CG*, the *DG* of *CG*, an *ADDG* of *DG* is called the *maximal ADDG* if adding any edge that in *DG* but not in *ADDG* to the *ADDG* will result in turn cycles in *CG*.

Lemma 1: Given a *CG* and a *DDG* of *CG*, if there is no cycle in the *DDG*, then no possible turn cycle can be formed in *CG* when the edges of *DDG* are the only available turns allowed at each node in *CG*.

Proof : Assume that there is a turn cycle *TC* = $(T_{d(\vec{e}_1),d(\vec{e}_2)}, T_{d(\vec{e}_2),d(\vec{e}_3)}, \ldots, T_{d(\vec{e}_k),d(\vec{e}_{k+1})})$ in *CG*. The turn cycle *TC* can be simply represented as *TC'*$(T_{d_1,d_2}, T_{d_2,d_3}, \ldots, T_{d_k,d_1})$ in a *DDG*, where $d_1 = d(\vec{e}_1) = d(\vec{e}_{k+1})$, $d_2 = d(\vec{e}_2)$, $d_3 = d(\vec{e}_3)$, …, and $d_k = d(\vec{e}_k)$. *TC'*$(T_{d_1,d_2}, T_{d_2,d_3}, \ldots, T_{d_k,d_1})$ is a cycle in the *DDG*. This contradicts the assumption. □

We now give examples to explain the above definitions. Figure 1(a) is an example of a switch-based network with irregular interconnection. The corresponding network topology of Figure 1(a) is shown in Figure 1(b). A *coordinated tree* for the network topology shown in Figure 1(b) is shown in Figure 1(c). In Figure 1(c), we have $Y(v_1) = 0$, $X(v_2) = 2$, and v_3 is the *right* node, *left* node, and *right-down* node of v_5, v_4, and v_1, respectively. For a link in Figure 1(b), if it is a link in Figure 1(c), then it is a tree link. Otherwise it is a cross link. The *CG* with respect to Figure 1(b) and Figure 1(c) is shown in Figure 1(d). In Figure 1(d), $d(<v_2,v_4>)$ = *RU_CROSS*, $d(<v_5,v_2>)$ = *RD_TREE*, T_{RD_TREE,RU_CROSS} is a *turn*, *TC* = $(T_{d(<v_5,v_1>),d(<v_1,v_3>)}, T_{d(<v_1,v_3>),d(<v_3,v_5>)}, T_{d(<v_3,v_5>),d(<v_5,v_1>)})$ is a turn cycle, and the set of direction *D* = {*LU_TREE*, *RD_TREE*, *LD_CROSS*, *RU_CROSS*, *R_CROSS*, and *L_CROSS*}. In Figure 1(d), the thick links are tree links and thin links are cross links. The corresponding *DG* of Figure 1(d) is shown in Figure 1(e). Figure 1(f) is an *ADDG* of Figure 1(d) and Figure 1(e). In Figure 1(f), the *ADDG* has two nodes, *LD_CROSS* and *RD_TREE*, and two edges (or *turns*), T_{LD_CROSS,RD_TREE} and T_{RD_TREE,LD_CROSS}. The two edges form a cycle in the *ADDG*. However, if each node of *CG* shown in Figure 1(d) only allows these two *turns*, no turn cycle can be formed in the *CG*. From Figure 1(f), we can see that a cycle in a *DDG* may not result in a turn cycle in a *CG*.

4. The *DOWN/UP Routing*

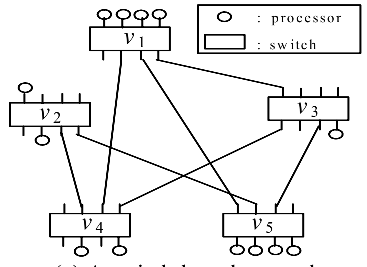
Given an irregular network topology $G = (V, E)$, the construction of the *DOWN/UP routing* consists of the following three phases.

Phase 1: Construct a $CT = (V, E')$ from *G* and the $CG = (V, \vec{E})$ with respect to *G* and *CT*.

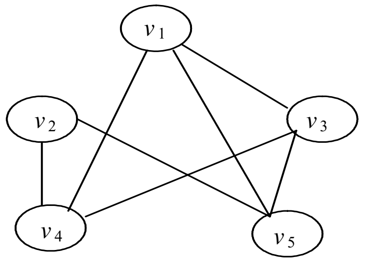
Phase 2: Find the *maximal ADDG max_ADDG* = $(D, \vec{T}')$ from the complete direction graph $CDG = (D, \vec{T})$.

Phase 3: Let $PT = \vec{T} - \vec{T}'$ be the set of prohibited turns for each node in *CG*. According to *CG*, remove the redundant prohibited turns for each node and find the possible shortest routing paths for any two nodes based the available turns of each node to form the *DOWN/UP routing*.

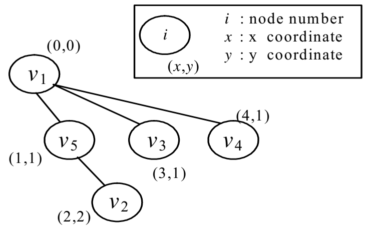
In the following, we explain each phase in details.



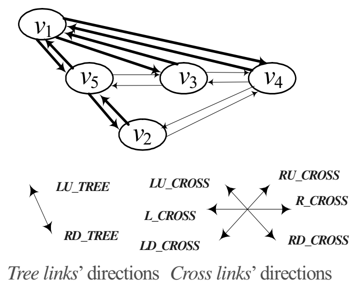
(a) A switch-based network.



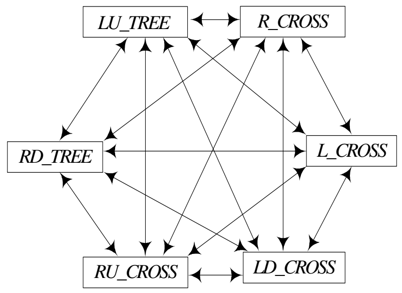
(b) The network topology of (a).



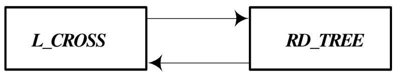
(c) A coordinated tree of (b).



(d) A *CG* of (b) and (c).



(e) The *DG* corresponding to (d).



(f) An *ADDG* of (d) and (e).

Figure 1. Examples for Definitions 1 to 11.

4.1. Phase 1

There are many ways to construct a coordinated tree from a network topology. Different coordinated trees lead to different performance of routing algorithms. How to construct a coordinated tree that leads to a better performance for routing algorithms is an important issue. One of the main contributions of this paper is that we propose a better way to construct a coordinated tree that leads to a better performance of routing algorithms. The construction of a coordinated tree $CT = (V, E')$ is composed of the following steps:

Step 1. Initially, let Q be an empty queue, *Visited* be an array of size $|V|$ initialized with 0, and E' be an empty set.

Step 2. Select v with the smallest node number from V. Set *Visited*$[v] = 1$ and insert v to Q.

Step 3. Delete v from Q.

Step 4. Let $W = \{w_1, w_2, \ldots, w_k\}$ be the set of nodes adjacent to v , where *Visited*$[w_i] = 0$, for $i = 1, \ldots, k$ and the node number of w_i is less than that of w_{i+1}. For $i = 1, \ldots, k$, insert w_i to Q, $E' = E' \cup \{(v , w_i)\}$, and set *Visited*$[w_i] = 1$.

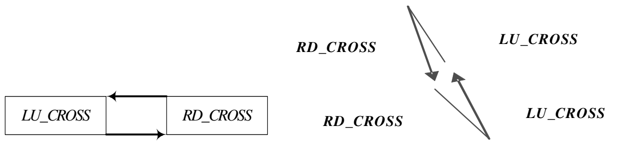
Step 5. If Q is not empty, perform Steps 3 and 4.

Step 6. Traverse the tree in preorder and nodes with smaller node numbers are visited first. Set the values of $X(v)$ and $Y(v)$ of node v as the order of preorder traversal and the level of the spanning tree, respectively.

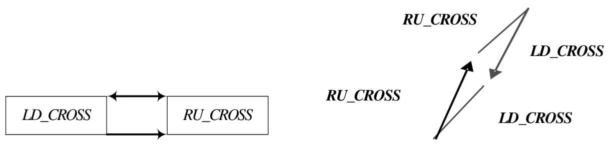
From Definition 5, the construction of the $CG = (V, \vec{E})$ with respect to $G = (V, E)$ and $CT = (V, E')$ is straightforward.

4.2. Phase 2

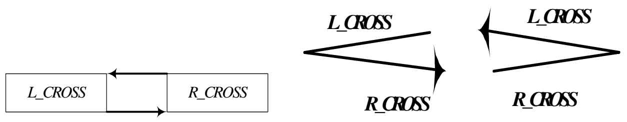
The complete direction graph contains all eight directions that can be defined in a *CG*, that is, it contains all possible turns that can be found in a *CG*. If we can find a maximal *ADDG* from the complete direction graph, then apply the edges (turns) in the maximal *ADDG* to each node of a *CG* will also result in no turn cycle. There are two issues to find the *maximal ADDG* from the complete direction graph. The first issue is to decide what edges should be removed (prohibited) from the complete direction graph. The second issue is the routing algorithm derived from the found maximal *ADDG* should perform efficiently. For the first issue, we use an incremental method to remove edges step by step from the complete direction graph to obtain a maximal *ADDG*. For the second issue, when removing edges from a *DDG* in each step, we will try to prevent the traffic from flowing to the root of a *CG* and push the traffic downward to the leaves of a *CG*. The process of finding the maximal *ADDG* from the complete direction graph consists of the following four steps:



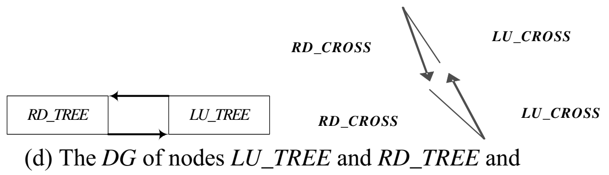
(a) The *DG* of nodes *LU_CROSS* and *RD_CROSS* and a possible turn cycle.



(b) The *DG* of nodes *LD_CROSS* and *RU_CROSS* and a possible turn cycle.



(c) The *DG* of nodes *L_CROSS* and *R_CROSS* and a possible turn cycle.



(d) The *DG* of nodes *LU_TREE* and *RD_TREE* and a possible turn cycle.

Figure 2. The *DGs* of node pairs and their corresponding possible turn cycles.

Step 1. Find the maximal *ADDGs* $ADDG_1$, $ADDG_2$, $ADDG_3$, and $ADDG_4$, from *DG*s of nodes *LU_CROSS* and *RD_CROSS*, nodes *LD_CROSS* and *RU_CROSS*, nodes *L_CROSS* and *R_CROSS*, and nodes *LU_TREE* and *RD_TREE* from the complete direction graph, respectively.

Step 2. Combine $ADDG_1$ with $ADDG_2$ by adding edges between nodes in $ADDG_1$ and $ADDG_2$ to form a new *DDG* and find a maximal *ADDG* $ADDG_5$ from the new formed *DDG*.

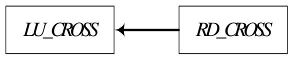
Step 3. Combine $ADDG_3$ with $ADDG_5$ by adding edges between nodes in $ADDG_3$ and $ADDG_5$ to form a new *DDG* and find a maximal *ADDG* $ADDG_6$ from the new formed *DDG*.

Step 4. Combine $ADDG_4$ with $ADDG_6$ by adding edges between nodes in $ADDG_4$ and $ADDG_6$ to form a new *DDG* and find a maximal *ADDG* $ADDG_7$ from the new formed *DDG*. The found $ADDG_7$ is a maximal *ADDG* of the complete direction graph.

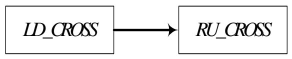
A. Step 1

In this step, we want to find the maximal *ADDGs* $ADDG_1$, $ADDG_2$, $ADDG_3$, and $ADDG_4$ from *DG*s of nodes *LU_CROSS* and *RD_CROSS*, nodes *LD_CROSS* and *RU_CROSS*, nodes *L_CROSS* and *R_CROSS*, and nodes *LU_TREE* and *RD_TREE* from the complete direction graph, respectively. The reason we choose these node pairs is that the *DG* of each node pair contains edges with opposite directions. These edges form a cycle that may lead to a turn cycle. Figure 2 shows the *DG*s of these node pairs and their corresponding possible turn cycles.

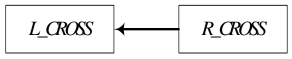
To find the maximal *ADDG* from the *DG* of each node pairs, we must remove one of the two edges of the *DG*. For each node *v* in a *CG*, the *LU_CROSS* and *RU_CROSS* directions indicate that the traffic flow is going upward from node *v* to other nodes whose *Y* coordinate is less than that of node *v*. The edges (*turns*) T_{RU_CROSS,LD_CROSS} and T_{LU_CROSS,RD_CROSS} in Figure 2(a) and Figure 2(b), respectively, will make the traffic flow goes upward before downward. In order to push the traffic flow downward to the leaves of the corresponding *CT*, we remove edges T_{RU_CROSS,LD_CROSS} and T_{LU_CROSS,RD_CROSS} from Figure 2(a) and Figure 2(b), respectively. The maximum *ADDG*s $ADDG_1$ and $ADDG_2$ for the *DG*s shown in Figure 2(a) and Figure 2(b) are shown in Figure 3(a) and Figure 3(b), respectively. For the cycle shown in Figure 2(c), since the removal of either edge leads to the same result, we randomly remove edge T_{L_CROSS,R_CROSS} from the *DG*. The $ADDG_3$ is obtained and is shown in Figure 3(c). For each node *v* in a *CG*, the *LU_TREE* direction indicates that the traffic is flowing from node *v* to the root of the corresponding *CT*. For the cycle shown in Figure 2(d), in order to prevent the traffic from flowing to the root of a corresponding *CT*, we remove edge T_{RD_TREE,LU_TREE} from the *DG*. The $ADDG_4$ is formed and is shown in Figure 3(d).



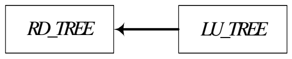
(a) $ADDG_1$



(b) $ADDG_2$



(c) $ADDG_3$



(d) $ADDG_4$

$$\begin{tikzcd} LU\_CROSS & RD\_CROSS \arrow[l] \end{tikzcd}$$

$$\begin{tikzcd} LD\_CROSS \arrow[r] & RU\_CROSS \end{tikzcd}$$

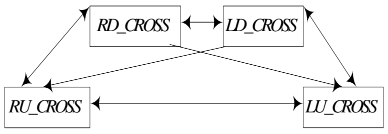
$$\begin{tikzcd} L\_CROSS & R\_CROSS \arrow[l] \end{tikzcd}$$

$$\begin{tikzcd} RD\_TREE & LU\_TREE \arrow[l] \end{tikzcd}$$

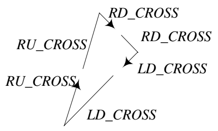
Figure 3. The maximal *ADDG*s of *DGs* shown in Figure 2.

B. Step 2

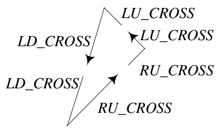
In this step, we want to combine $ADDG_1$ with $ADDG_2$ by adding edges between nodes in $ADDG_1$ and $ADDG_2$ to



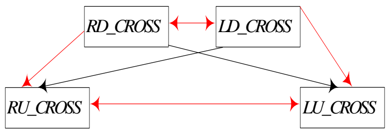
(a) The *DDG* of combining $ADDG_1$ with $ADDG_2$.



(b) A possible *turn cycle* for C_1.

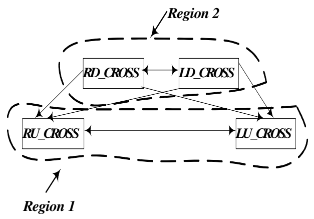


(c) A possible *turn cycle* for C_2.

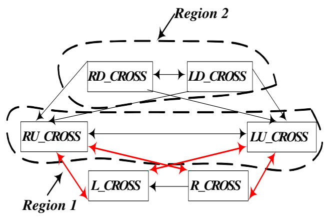


(d) $ADDG_5$.

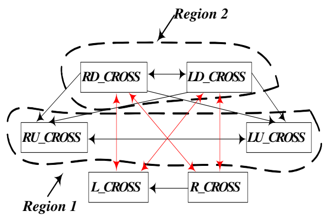
Figure 4. Combine $ADDG_1$ with $ADDG_2$ to form $ADDG_5$.



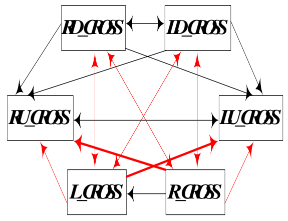
(a) Two regions for $ADDG_5$.



(b) Combine $ADDG_3$ with *Region* 1.



(c) Combine $ADDG_3$ with *Region* 2.



(d) $ADDG_6$.

Figure 5. Combine $ADDG_3$ with $ADDG_5$ to form $ADDG_6$.

form a new *DDG* and find a maximal *ADDG* $ADDG_5$ from the new formed *DDG*.

The *DDG* of combining $ADDG_1$ with $ADDG_2$ is shown in Figure 4(a). When adding edges between node *RD-CROSS* in $ADDG_1$ and nodes in $ADDG_2$, a cycle $C_1(T_{RU\_CROSS,RD\_CROSS}, T_{RD\_CROSS,LD\_CROSS}, T_{LD\_CROSS,RU\_CROSS})$ is formed. This cycle may form the turn cycle, shown in Figure 4(b), in a *CG*. When adding edges between node *LU_CROSS* in $ADDG_1$ and nodes in $ADDG_2$, a cycle $C_2(T_{LU\_CROSS,LD\_CROSS}, T_{LD\_CROSS,RU\_CROSS}, T_{RU\_CROSS,LU\_CROSS})$ is formed. This cycle may form the turn cycle, shown in Figure 4(c), in a *CG*. Since the edges (*turns*) T_{RU_CROSS,RD_CROSS} and T_{LU_CROSS,LD_CROSS} in cycles C_1 and C_2 will make the traffic flow goes upward before downward, respectively. In order to push the traffic flow downward the leaves of a corresponding *CT*, we remove edges T_{RU_CROSS,RD_CROSS} and T_{LU_CROSS,LD_CROSS} from the *ADD* shown in Figure 4(a). The $ADDG_5$ is formed and is shown in Figure 4(d).

C. Step 3

In this step, we want to combine $ADDG_3$ with $ADDG_5$ by adding edges between nodes in $ADDG_3$ and $ADDG_5$ to form a new *DDG* and find a maximal *ADDG* $ADDG_6$ from the new formed *DDG*. For nodes in Figure 4(d), we have the following observations:

Observation 1: Any combination of edges (*turns*) from nodes *LD_CROSS* and *RD_CROSS* would not have upward directions in a *CG*.

Observation 2: Any combination of edges (*turns*) from nodes *LU_CROSS* and *RU_CROSS* would not have downward directions in a *CG*.

Therefore, we divide $ADDG_5$ into *Region* 1 and *Region* 2 as shown in Figure 5(a). For the $ADDG_3$ shown in Figure 3(c), edge T_{R_CROSS,L_CROSS} indicates that the traffic is flowing between nodes in the same level of a corresponding *CT*. To combine the $ADDG_3$ with *Region* 1 or *Region* 2 shown in Figure 5(a), we have the following observations:

Observation 3: If we combine $ADDG_3$ with *Region* 1 to form a *DDG* shown in Figure 5(b), no turn cycles can be formed from the *DDG* shown in Figure 5(b).

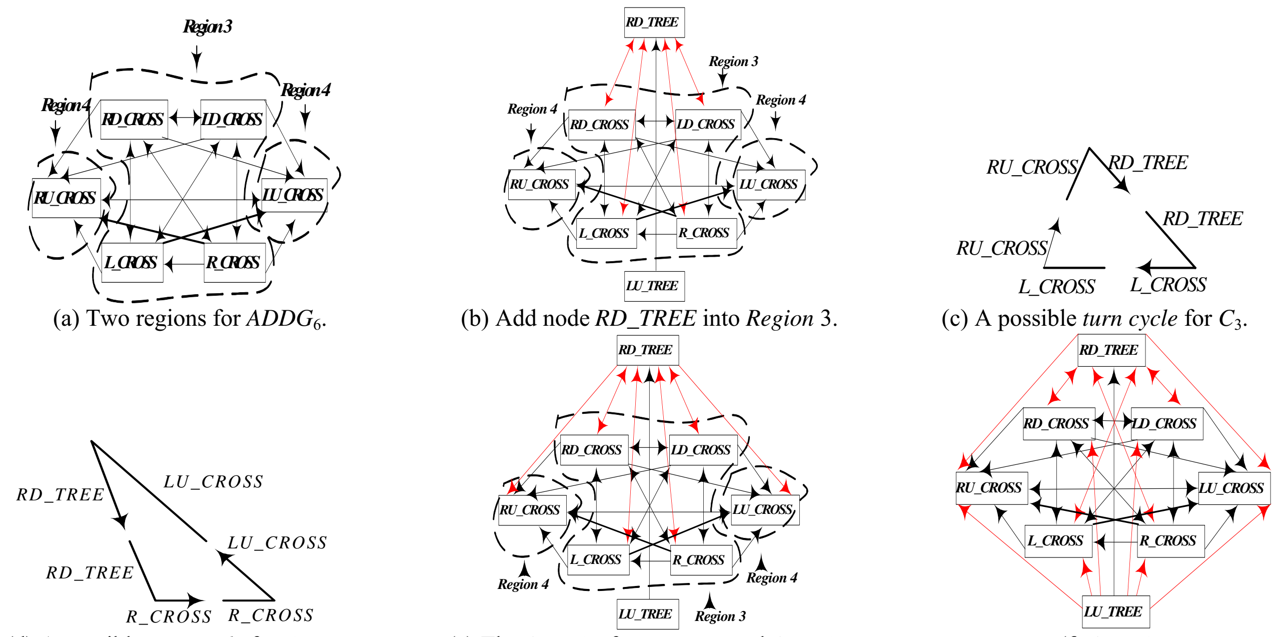
Observation 4: If we combine $ADDG_3$ with *Region* 2 to form a *DDG* shown in Figure 5(c), no turn cycles can be formed from the *DDG* shown in Figure 5(c).

Observation 5: If we combine $ADDG_3$ with $ADDG_5$, a possible turn cycle can be formed from node *v* in *Region* 1 to nodes in $ADDG_3$, *Region* 2, and back to node *v*.

To combine $ADDG_3$ with $ADDG_5$ to form $ADDG_6$, based on Observations 3, 4, and 5, we can first combine Figure 5(b) and Figure 5(c). Then by removing either edges from nodes in *Region* 1 to nodes in $ADDG_3$ or edges from nodes in *Region* 2 to nodes in $ADDG_3$, we can obtain $ADDG_6$. Since any combination of edges (turns) between nodes in $ADDG_3$ and nodes in *Region* 1 would not have downward directions in a *CG*, to keep the traffic flow downward, we remove edges from nodes in *Region* 1 to nodes in $ADDG_3$. We then obtain $ADDG_6$ as shown in Figure 5(d).

D. Step 4

In this step, we want to combine $ADDG_4$ with $ADDG_6$ by adding edges between nodes in $ADDG_4$ and $ADDG_6$ to form a new *DDG* and find a maximal *ADDG* $ADDG_7$



(a) Two regions for *ADDG*$_6$. (b) Add node *RD_TREE* into *Region* 3. (c) A possible *turn cycle* for C_3.

(d) A possible *turn cycle* for C_4. (e) The *ADDG* of *RD_TREE* and *ADDG*$_6$. (f) *ADDG*$_7$.

Figure 6. Obtain a maximal *ADDG* for the complete direction graph.

from the new formed *DDG*. The found *ADDG*$_7$ is a maximal *ADDG* of the complete direction graph.

For nodes in Figure 5(d), we have Observation 2 and the following observation:

Observation 6: Any combination of edges (turns) from nodes *LD_CROSS* and *RD_CROSS*, *L_CROS*, and *R_CROSS* would not have upward directions in a *CG*.

Therefore, based on Observations 2 and 6, we divide *ADDG*$_6$ into *Region* 3 and *Region* 4 as shown in Figure 6(a).

When adding edges between node *RD_TREE* and nodes in *Region* 3, no turn cycles can be formed from the *DDG* and any combination of edges (turns) from the *DDG* would not have upward directions in a *CG*, that is, the turns defined in the *DDG* will push the traffic downward to the leaves of a corresponding *CT*. We keep all edges in between node *RD_TREE* and nodes in *Region* 3 and the *DDG* is shown in Figure 6(b).

Based on Figure 6(b), when adding edges between node *RD_TREE* and nodes in *Region* 4, two cycles $C_3(T_{RD\_TREE,L\_CROSS}$, T_{L_CROSS,RU_CROSS}, $T_{RU\_CROSS,RD\_TREE})$ and $C_4(T_{RD\_TREE,R\_CROSS}$, T_{R_CROSS,LU_CROSS}, $T_{LU\_CROSS,RD\_TREE})$ are formed. These cycles may form *turn cycles* as shown in Figure 6(c) and Figure 6(d), respectively. Therefore, we remove the two edges T_{LU_CROSS,RD_TREE} and T_{RU_CROSS,RD_TREE} and the *DDG* is shown in Figure 6(e).

When adding edges between node *LU_TREE* and nodes in *ADDG*$_6$ based on Figure 6(e), a possible turn cycle can be formed from node *RD_TREE* to nodes in *ADDG*$_6$, *LU_TREE*, and back to node *RD_TREE*. For each node *v* in a *CG*, the *LU_TREE* direction indicates that the traffic is flowing from node *v* to the root of the corresponding coordinated tree. To prevent the traffic from flowing to the root of a corresponding *CT*, we remove all edges from nodes in *ADDG*$_6$ to node *LU_TREE* to form an *ADDG* as shown in Figure 6(f). This new formed *ADDG* is *ADDG*$_7$, a maxima*l ADDG* of the complete direction graph.

4.3. Phase 3

In Phase 2, we have removed eighteen edges from the complete direction graph. These eighteen edges form a set of prohibited *turns* *PT* = $\{T_{RD\_TREE,LU\_TREE}$, T_{RD_CROSS,LU_TREE}, T_{L_CROSS,LU_TREE}, T_{R_CROSS,LU_TREE}, T_{RU_CROSS,LU_TREE}, T_{LU_CROSS,LU_TREE}, T_{LD_CROSS,LU_TREE}, T_{RU_CROSS,LD_CROSS}, T_{RU_CROSS,RD_CROSS}, T_{LU_CROSS,LD_CROSS}, T_{LU_CROSS,RD_CROSS}, T_{LU_CROSS,RD_TREE}, T_{RU_CROSS,RD_TREE}, T_{L_CROSS,R_CROSS}, T_{R_CROSS,RU_CROSS}, T_{R_CROSS,LU_CROSS}, T_{L_CROSS,RU_CROSS}, $T_{L\_CROSS,LU\_CROSS}\}$. Since the *ADDG*$_7$ found in Phase 2 is a maximal *ADDG* of the complete direction graph, when the prohibited turns in *PT* applied to nodes of a *CG*, it is possible that some prohibited turns are redundant for some nodes. For example, for the *CG* shown in Figure 7, in which the thick lines are the tree links and the thin lines are cross links, the turn from node v_9 through node v_6 to node v_8 (T_{RU_CROSS,RD_TREE}) and the turn from node v_4 through node v_{11} to node v_5 (T_{LU_CROSS,RD_TREE}) are unnecessary prohibited turns for nodes v_6 and v_{11}, respectively, since these prohibited turns on these nodes do not form turn cycles in the *CG*. In this phase, we want to remove these redundant prohibited turns for each node in *CG* from *PT* and find routing paths from the available turns of each node in *CG* to form the *DOWN/UP routing*.

We propose a *cycle detection* algorithm, which is similar to that in [4], to remove the redundant prohibited turns for each node in *CG* from *PT*. In the cycle detection algorithm, we only consider to release the prohibited turns T_{LU_CROSS,RD_TREE} and T_{RU_CROSS,RD_TREE} in each node. The reasons are two-fold. First, only the prohibited turns T_{LU_CROSS,RD_TREE} and T_{RU_CROSS,RD_TREE} in *PT* can help the traffic flow not go upward to the root and push the traffic flow downward to the leaves of a corresponding *CT*. Second, each node in a *CG*, except the leaves of a corresponding *CT*, has the output channel with direction *RD_TREE*. The number of T_{LU_CROSS,RD_TREE} and T_{RU_CROSS,RD_TREE} may be more than that of other prohibited turns in a *CG*. The algorithm is given as follows:

```
Algorithm cycle_detection(CG, PT)   /* CG = (V, $\vec{E}$) */
1. Let in(v) be the set of input channels of node v in V
   whose direction is LU_CROSS or RU_CROSS.
2. Let out(v) be the set of output channels of node v in V
   whose direction is RD_TREE.
3. $\forall v \in V$, for each pair of ($\vec{e}_1, \vec{e}_2$) do /* where $\vec{e}_1 \in in(v)$
   and $\vec{e}_2 \in out(v)$ */
4. { Initialize stacks S and D.
5. Starting from v, visit its adjacency node v' through $\vec{e}_2$
     and mark $\vec{e}_2$.
6.    Let in_channel = $\vec{e}_2$ and push $\vec{e}_2$ into D.
7.    if (node v' has an output channel $\vec{e}'$ = <v', v"> that
          is not marked and does not form a prohibited
          turn with d(in_channel)) then { Push v" into S,
          push $\vec{e}'$ into D, mark $\vec{e}'$, and v' = v". }
      else if ($S \neq \emptyset$) then { v' = pop(S), in_channel =
          pop(D). }
      else { Release the turn $T_{d(c_1),d(c_2)}$ of node v.
       Check next input and output channel pair of node v. }
8.    if (v' = v and in_channel = $\vec{e}_1$)
        then {A turn cycle exists and the prohibited turn in
              node v can not be released.
              Check next input and output channel pair of
              node v. }
9.    Goto line 7.}
  end_of_cycle_detection
```

Algorithm *cycle_detection* uses the depth first search (DFS) to check whether a prohibited turn of a node can be released or not. The time complexity of algorithm *cycle_detection* is $O(d*|V|^2)$, where *d* is the degree of each node in a *CG*. After algorithm *cycle_detection* is performed, the prohibited turns for each node are determined. We then can use the shortest path algorithms based on the prohibited turns to find the routing path between any two nodes. We call the routing algorithm by applying the proposed method as *DOWN/UP routing* since the packet must go downward cross links then go upward cross links.

Theorem 1: The *DOWN/UP* routing is deadlock-free and connectivity between each node pair is guaranteed.

Proof: Based on Phase 2, there is at least one prohibited turn to break each turn cycle in the maximal *DG*. Therefore, this routing algorithm is deadlock-free. Since the *turn* T_{LU_TREE,RD_TREE} is not prohibited for each node in a *CG*, each flit from any source node to it destination node can first go upward their least common ancestor and then goes downward to the destination node. Therefore, connectivity between each node pair is guaranteed. □

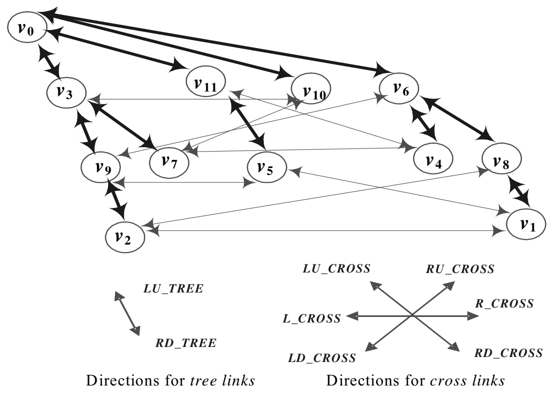
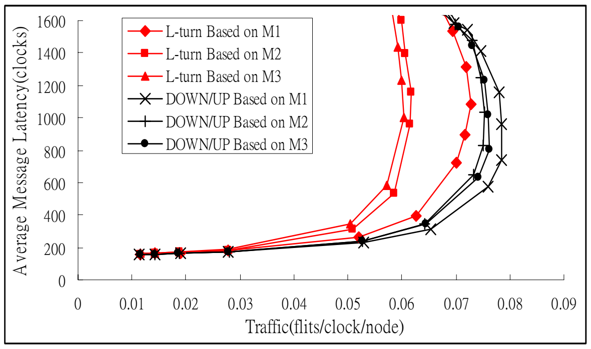


Figure 7. An example for redundant prohibited turns.

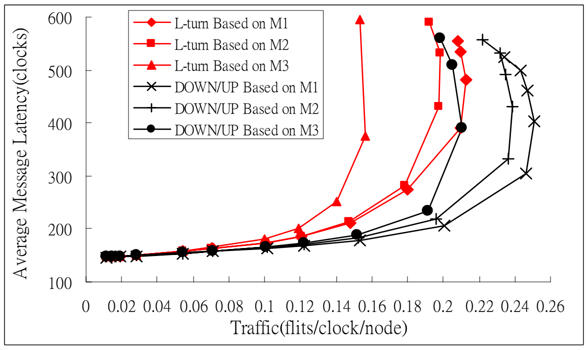
5. Performance Analyses

To evaluate the performance of the *DOWN/UP routing*, we have implemented the proposed method along with the *L-turn routing* on the IRFlexSim0.5 simulator [2], a wormhole technique simulator written in C. To simulate an irregular network, we assume that the network contains 128 switches. Each switch is associated with a processor. The port number of each switch is set to 4 and 8. Each port has an input and an output channels. The packet length is set to 128 flits. The delay of a flit passes through a link is one clock. The delay of a routing header to be routed and arbitrated to the output channel is one clock. A data flit to be transmitted from the input channel to output channel is one clock. A uniform traffic pattern is assumed.

For both 4-port and 8-port configurations, we have randomly generated 10 irregular networks each as test



(a) 4-port configuration.



(b) 8-port configuration.

Figure 8. The average message latency and accepted traffic.

samples. Since different coordinated trees will lead to different performance, for each test sample, we used three coordinated trees, $M1$, $M2$, and $M3$, to evaluate the performance of the *L-turn routing* and the *DOWN/UP routing*. When perform the preorder traversal to determine the x-axis coordinate of a node in a coordinated tree, the next node to be traversed for $M1$, $M2$, and $M3$ is the node with the smallest node number, a randomly selected one, and the node with the largest node number, respectively. The method for $M1$ is the one we proposed in Phase 1 of Section 4. When construct a coordinated tree, we choose the node with the smallest node number as the root a spanning tree. Since the *L-turn routing* and the *DOWN/UP routing* are both non-minimal adaptive deadlock-free routing algorithms, in the simulation, we use the shortest possible paths between all pairs of source and destination nodes to transmit flits. For any two nodes, it is possible that more than one shortest possible path exist. For this case, one of them is selected randomly.

Figure 8(a) and Figure 8(b) show the average simulation results of message latency and accepted traffic for 4-port and 8-port configuration, respectively. The message latency is defined as the time elapsed since the packet transmission is initiated at a node until the packet is received at the destination node. The smaller the message latency, the less the flits delay. The throughput is defined as the accepted traffic measured in flits/clock pernode (flits/clock/node). The higher the throughput, the better the bandwidth offered. From Figure 8, for the same coordinated tree and the same configuration, we observe that the proposed routing algorithm has smaller message latency and higher throughput than that of the *L-turn routing*. For different coordinated trees and the same configuration, we observe that both routing algorithms have the smallest message latency and the highest throughput if $M1$ is used.

In order to analyze the characteristics of both routing algorithms, for each test sample, we also measure the *node utilization*, the *traffic load*, the *degree of hot spots*, and the *leaves utilization* when both routing algorithms reach their maximal throughputs. Table 1 shows the average simulation results of node utilization for both 4-port and 8-port configurations. The utilization of an output channel of a node is defined as the average number of flits across the node through the output channel during one clock. The node utilization of a node is defined as the sum of utilization of all output channels of the node divided by the number of ports connecting to other switches. The higher the node utilization is, the smoother the traffic flows. From Table 1, for the same coordinated tree and the same configuration, we observe that the node utilization of the proposed routing algorithm is higher than that of the *L-turn routing*. For different coordinated trees and the same configuration, both routing algorithms have the best node utilization if $M1$ is used.

Table 1. The average simulation results of node utilization.

	L-turn routing		*DOWN/UP routing*	
	4-port	8-port	4-port	8-port
$M1$	0.115772	0.123159	0.123295	0.147124
$M2$	0.108101	0.111653	0.121793	0.139588
$M3$	0.095841	0.092198	0.120955	0.126071

Table 2 shows the average simulation results of traffic load for both 4-port and 8-port configurations. The traffic load is defined as the standard deviation of the node utilization of all nodes. The smaller the traffic load, the better the balanced traffic load. From Table 2, for the same coordinated tree and the same configuration, we observe that the traffic load of the proposed routing algorithm is less than that of *L-turn routing*. This indicates that the *DOWN/UP* routing has a more balanced traffic load than the *L-turn routing*. For different coordinated trees and the same configuration, both routing algorithms have the smallest traffic load if $M1$ is used.

Table 2. The average simulation results of traffic load.

	L-turn routing		*DOWN/UP routing*	
	4-port	8-port	4-port	8-port
*M*1	0.078314	0.048727	0.077657	0.043990
*M*2	0.081115	0.050460	0.078501	0.047316
*M*3	0.083969	0.053392	0.078047	0.049796

Table 3 shows the average simulation results of degree of hot spots. The degree of hot spots is defined as the percentage of the node utilization of nodes in levels 0 and 1 of a coordinated tree. The smaller the degree of hot spots is, the less the traffic congests. Table 4 shows the average simulation results of leave utilization. The leave utilization is defined as the average of node utilization of leaves of a coordinated tree. The more the leave utilization, the higher the traffic flow to leaves. From Tables 3 and 4, for the same coordinated tree and the same configuration, we observe that the proposed routing algorithm has less degree of hot spots and higher leave utilization, respectively. For different coordinated trees and the same configuration, we observe that both routing algorithms have the smallest degree of hot spots and the highest leave utilization if *M*1 is used.

Table 3. The average simulation results of degree of hot spots.

	L-turn routing		*DOWN/UP routing*	
	4-port	8-port	4-port	8-port
*M*1	12.85 %	13.26 %	12.00 %	9.930 %
*M*2	14.15 %	14.90 %	12.13 %	10.56 %
*M*3	16.18 %	18.43 %	12.16 %	11.25 %

Table 4. The average simulation results of leave utilization.

	L-turn routing		*DOWN/UP routing*	
	4-port	8-port	4-port	8-port
*M*1	0.07336	0.1065	0.082897	0.13807
*M*2	0.063953	0.093437	0.080773	0.131578
*M*3	0.050633	0.072627	0.078453	0.111609

From Tables 3 and 4, we can see that the *DOWN/UP routing* has less degree of hot spots and better leaves utilization than the *L-turn routing*. These lead to higher node utilization and more balanced traffic load for the *DOWN/UP routing* compared to the *L-turn routing*. That is why the *DOWN/UP routing* has smaller message latency and higher throughput than that of the *L-turn routing*.

6. Conclusion Remarks

From the simulation results shown in Section 5, we have the following remarks for the proposed routing algorithm and the *L-turn routing*.

Remark 1. The way we construct a coordinated tree (*M*1) leads to the best performance for both *DOWN/UP routing* and *L-turn routing* compare to other methods (*M*2 and *M*3).

Remark 2. Since traffic in the *DOWN/UP routing* flows more downward to the leaves of a coordinated tree than that of the *L-turn routing.* Under the same coordinated tree and the same configuration, the *DOWN/UP routing* outperforms the *L-turn routing* in terms of node utilization, traffic load, the degree of hot spots, leaves utilization, message latency, and throughput for all test samples.

Acknowledgments

This work in this paper was partially supported by Program for Promoting Academic Excellence of Universities under contract 89-E-FA04-1-4.

References

[1] C. J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing,' *J. ACM,* Vol. 41, No. 5, pp. 874-902, Sept. 1994.

[2] IRFlexSim0.5 is available in" http://www.usc.edu/dept/ceng/pinkston/tools.html"

[3] A. Jouraku, A. Funahashi, H. Amano, and M. Koibuchi, "Routing Algorithms on 2D Turn Model for Irregular Networks," *the Sixth International Symposium on Parallel Architectures, Algorithms, and Networks(I-SPAN),* pp.289-294, May. 2002

[4] A. Jouraku, A. Funahashi, H. Amano, and M. Koibuchi,"L-turn routing: An Adaptive Routing in Irregular Networks," *the International Conference on Parallel Processing,* pp.374-383, Sep. 2001.

[5] G. Pfister and V. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Trans. Computers.* C34 (10):943-948, Oct. 1985.

[6] A. Robles , J. Duato, and J.C. Sancho," A Flexible Routing Scheme for Networks of Workstations," *ISHPC*, pp. 260-267,2000.

[7] M. D. Schroeder et al., "Autonet: A High-Speed Self-Configuring Local Area Network Using Point-to-Point Links," *SRC research report 59,* DEC, Apr. 1990.

[8] F. Silla and J. Duato, "High-Performance Routing in Networks of Workstations with Irregular Topology," *IEEE Transactions on Parallel and Distributed Systems,* Vol.11, No.7, pp. 699-719, July 2000.

IEEE COMPUTER SOCIETY