Mapping and Scheduling for Circuit-Switched Network-on-Chip Architecture

Chia-Ming Wu, Hsin-Chou Chi, and Ruay-Shiung Chang

Network-on-chip (NoC) architecture provides a high-performance communication infrastructure for system-on-chip designs. Circuit-switched networks guarantee transmission latency and throughput; hence, they are suitable for NoC architecture with real-time traffic. In this paper, we propose an efficient integrated scheme which automatically maps application tasks onto NoC tiles, establishes communication circuits, and allocates a proper bandwidth for each circuit. Simulation results show that the average waiting times of packets in a switch in 6×6, 8×8, and 10×10 mesh NoC networks are 0.59, 0.62, and 0.61, respectively. The latency of circuits is significantly decreased. Furthermore, the buffer of a switch in NoC only needs to accommodate the data of one time slot. The cost of the switch in the circuit-switched network can be reduced using our scheme. Our design provides an effective solution for a critical step in NoC design.

Keywords: Network-on-chip architecture, circuit-switched networks, mapping, scheduling.

Manuscript received Oct. 17, 2008; revised Dec. 9, 2008; accepted Dec. 22, 2008.

Chia-Ming Wu (phone: +886 3 8632722, email: cmwu@mail.ndhu.edu.tw), Hsin-Chou Chi (email: hcchi@mail.ndhu.edu.tw), and Ruay-Shiung Chang (email: rschang@mail.ndhu.edu.tw) are with the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan.

I. Introduction

Network-on-chip (NoC) architecture decouples computation from communication to overcome the communication problems in system-on-chip (SoC) design. NoC is a tile-based architecture where each tile contains an intellectual property (IP) block, such as a processor, digital signal processing (DSP) unit, field-programmable gate array (FPGA) block, or embedded memory. A routing switch is embedded within each tile to deliver communication packets between tiles. The resource network interface (RNI) provides a well-defined interface to connect an IP and its associated routing switch. Due to the structured design, the electrical parameters of the links in the tile-based architecture can be well controlled and optimized. Such on-chip interconnection networks thus provide a high-performance chip-level communication infrastructure with regularity and modularity [1]-[6].

Unlike dedicated wires and shared buses, links in an NoC are shared by communication circuits. The communication requirement of each circuit has to be satisfied so that the system based on the NoC can properly work. An application running on an embedded system can be represented by a communication task graph (CTG) such as the one shown in Fig. 1. In this graph, the vertex stands for an IP or computation task, while the edge stands for communication flow between IPs/tasks [7], [8].

A mapping algorithm is used to assign IPs in the CTG to tiles of the NoC. Figure 1 shows a mapping example. The mapping algorithm profoundly affects the performance and power consumption of communication in the NoC. Another critical issue in communication performance is the switch. The switch has to efficiently schedule and forward communication packets to ensure that the guaranteed communication service is

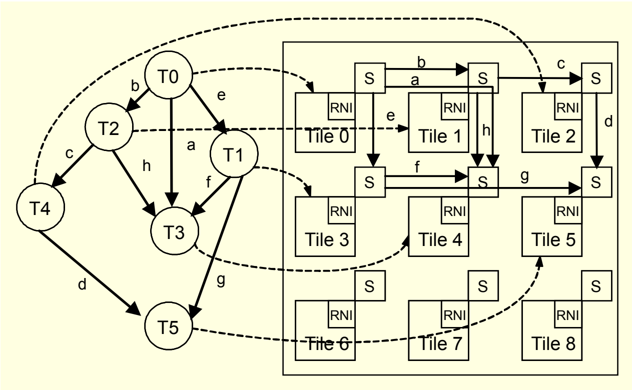

Fig. 1. Example of mapping from a CTG to an NoC. The solid lines in the NoC represent the circuits reserved by the mapping algorithm.

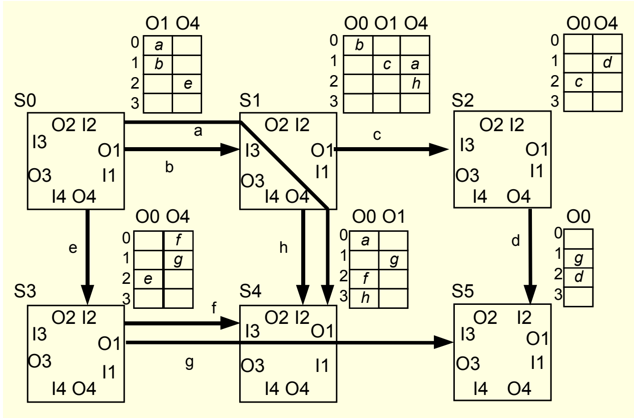

Fig. 2. Time slot assignment of circuits shown in Fig. 1.

satisfied [7]-[10].

A circuit-switched network can provide guaranteed transmission latency and throughput in an NoC; therefore, it is particularly suitable for supporting real-time traffic [11]-[13]. A problem in circuit-switched networks is the circuit arrangement. Using the time division multiplexing (TDM) technique, multiple circuits can be multiplexed through the link between two nodes. The bandwidth of a link can be divided into a certain number of time slots in a periodical frame. A scheduling algorithm is needed to arrange connections in switches and allocate proper time slots to circuits. The scheduling algorithm has a significant impact on the latency of the communication in the NoC. Figure 2 shows the time slot assignment of the circuits shown in Fig. 1. There are eight circuits in this example.

In Fig. 2, the table associated with each switch indicates the output time slot reservation for the circuits through the switch. For example, the table of switch S1 has three columns for outputs O0, O1, and O4. Two circuits, **a** and **b**, enter this switch at inputs I0 and I3. Three circuits, **a**, **c**, and **h**, depart at outputs O0, O1, and O4. Output O0 and input I0 are the local output port and input port of the switch, respectively. The switch uses these two ports to communicate with the local tile. We omit these two ports in the figure for simplicity. An off-line scheduling algorithm assigns time slot 0 of output O0 to circuit **b** (I3). It also assigns time slot 1 of output O1 to circuit **c** (I0). Time slots 1 and 2 of output O4 are assigned to circuits **a** (I3) and **h** (I0). If the delivery of a packet by the switch needs one clock cycle and the arrival time of a packet from circuit **a** is in time slot 0, the data from circuit **a** does not need to be stored in the buffer in the switch. This reduces the memory requirements of the switch; thus, a good scheduling algorithm can also decrease the cost of the NoC.

In this paper, we propose an integrated mapping and scheduling scheme for on-chip networks. This algorithm can map an application onto the NoC architecture. Furthermore, it can allocate and schedule time slots for the switches in order to minimize the latency of the circuits in the network. Our design provides an effective mapping and scheduling solution for the circuit-switched NoC design. In our design, the integrated mapping and scheduling scheme is realized during the initialization stage of an NoC-based system by software.

The remainder of the paper is organized as follows. The next section describes our problem model. Our mapping and scheduling scheme is presented in section III. Section IV reports the experimental results for our algorithm. Finally, our work is summarized and concluded in section V.

II. Problem Model

There are basically two problems in NoC design, namely, mapping and scheduling. In the mapping of IPs, IPs are selected from the CTG and are assigned to the tiles of the NoC architecture. The mapping must at least satisfy the bandwidth requirement of each circuit. In scheduling, time slots are allocated to the switches so as to minimize the latency of the circuits in the network.

Definition 1. A communication task graph is a weighted digraph CTG(U, E), where U is a set of IPs, and $E = \{(u_i, u_j) \mid u_i \in U, u_j \in U\}$ is a set of communications. For each $e_{i,j} = (u_i, u_j)$ in E, there is a weight $w_{i,j}$, which represents the requirement of the communication bandwidth from u_i to u_j.

Definition 2. An NoC architecture graph is a digraph NAG(V, L), where V is a set of NoC tiles, and $L = \{(v_i, v_j) \mid v_i \in V, v_j \in V\}$ is a set of direct links. For each $l_{i,j} = (v_i, v_j)$ in L, there is a weight $b_{i,j}$, which represents the bandwidth capacity of the link from v_i to v_j.

A circuit $p_{i,j}$ from v_i to v_j may pass through many tiles, and a link in an NAG may be in many circuits simultaneously. For a link $l_{i,j}$ in L, let $r_{i,j}$ denote the total required bandwidth of the communication flows through this link.

The function of mapping the communication task graph CTG(U, E) to the NoC graph NAG(V, L) is one-to-one and defined by $MAP(U)$：

$$map: U \rightarrow V \Rightarrow v_j = map(u_i), \forall u_i \in U, \exists v_j \in V,$$
$$\text{such that } r_{i,j} \leq b_{i,j}, \forall i, j.$$

The mapping is defined only when $|U| \leq |V|$.

Latency is defined as the duration it takes for a packet to be transported from the sender to the receiver. The major factors are the latency of the switch and latency of the link, which are denoted by L_s and L_l, respectively. The latency of a circuit $p_{i,j}$ for sending one bit of data from tile v_i to tile v_j can be analytically calculated as

$$L_{p_{i,j}} = \sum_{S_k \in p_{i,j}} L_{S_k} + \left(n_{\text{hops}_{i,j}} - 1\right) \times L_l, \tag{1}$$

where $n_{\text{hops}_{i,j}}$ is the number of switches the bit passes on its way from tile v_i to tile v_j. The latency L_l is the latency of the link, L_{S_k} denotes the latency of the switch S_k in the circuit, and $n_{\text{hops}_{i,j}}$ depends on the routing policy of the network. According to the XY routing algorithm [14], the number of hops from v_i to v_j can be calculated as

$$n_{hops_{i,j}} = \left|x_{v_i} - x_{v_j}\right| + \left|y_{v_i} - y_{v_j}\right| + 1, \tag{2}$$

where x_{v_i} and x_{v_j} are the x-axis coordinates of v_i and v_j, and y_{v_i} and y_{v_j} are the y-axis coordinates of v_i and v_j.

In order to decrease $L_{p_{i,j}}$, the mapping algorithm has to minimize $n_{\text{hops}_{i,j}}$ for every $p_{i,j} \in P$.

The latency of a switch can be calculated as

$$L_{S_k} = T_{\text{transfer}} + T_{\text{waiting}}, \tag{3}$$

where T_{transfer} is the transfer time of a switch. This time is the period that a switch forwards a bit without storing this bit into the buffer. This time depends on the hardware design. The waiting time T_{waiting} is defined as the duration for a bit stored in buffer. This time depends on the scheduling algorithm. If each arriving bit can be delivered immediately by a proper output, T_{waiting} is zero and the switch does not need buffers. However, due to conflicts between circuits, such an arrangement is difficult to achieve, and buffers are typically required. In a mapped NoC, the goal of a scheduling algorithm is to minimize the total latency experienced in every switch.

III. Mapping and Scheduling

The main algorithm, shown in Fig. 3, is composed of two parts, mapping and scheduling. Step 2 is for mapping. We use a heuristic algorithm to find a mapping graph NAG(V, L). Then, the scheduling algorithm, comprising steps 3, 4, and 5, uses the mapping result to schedule the circuits. If the result of the scheduling cannot satisfy the requirement of the communication latency, the mapping algorithm is revoked to find another mapping.

The procedure is repeated until all communication latencies are satisfied. We summarize the major steps of the algorithm in Fig. 3.

Main (CTG, constraints, OTSTs):
Step 1. Input the CTG and latency constraints.
Step 2. Mapping.
Step 3. Decomposition.
Step 4. Time slot allocation.
Step 5. Latency minimization.
Step 6. If the latency is not satisfied, go to step 2.
Step 7. Output the output time slot tables (OTSTs).

Fig. 3. Main mapping and scheduling algorithm.

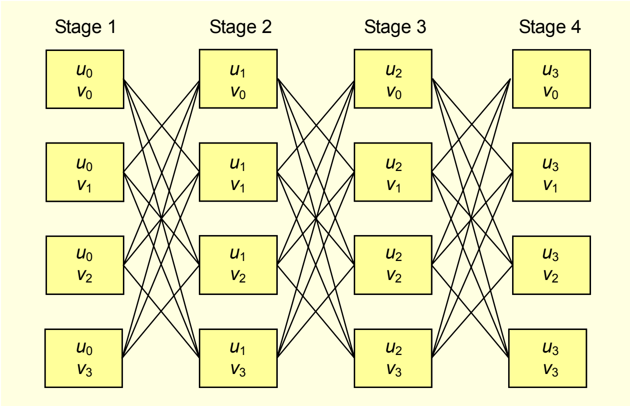

Fig. 4. Mapping problem viewed as a searching problem in a multi-stage graph.

1. Mapping Algorithm

The mapping problem can be viewed as a searching problem in a multistage graph. In Fig. 4, we show an example of a four-IP mapping problem. Each vertex in the multistage graph represents that an IP is assigned to a tile in the NoC architecture. A path that does not visit the same tile more than once in the multistage graph represents a whole mapping result. Such a path also needs to satisfy design constraints given by certain communication requirements, namely, maximum link bandwidth and circuit latency.

Figure 5 shows an example of the searching tree in the multistage graph for mapping a CTG(U, E) with four vertices onto a 2×2 NAG(V, L). The root node indicates that no

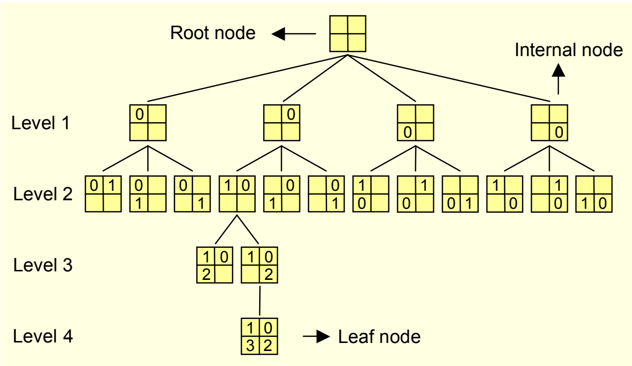

Fig. 5. Example of the mapping tree in the mapping process.

Mapping (CTG(U, E), constraints, NAG(V, L)):

Step 1. Set initial value ∞ to $v_i, \forall v_i \in V$.

Step 2. Select the IPs whose communication latencies are under constraint.

Step 3. Sort the remaining $u_i \in U$ according to $\sum_j e_{i,j}$. Store the result in the unmapped stack in decreasing order. The smallest is placed at the bottom of the stack. Push the IPs selected in step 2 into the unmapped stack.

Step 4. Pop an IP u from the unmapped stack.

Step 5. Search for an available tile $v \in V$ and assign this IP u to it. If there is not any proper tile $v \in V$, push this IP into the unmapped stack and pop an IP u from the mapped stack.

Step 6. Establish all related $p \in P$ in the NAG for this v.

Step 7. Calculate all r_{ij} and n_{hops}, denoted in (1) and (2).

Step 8. If there is any p unable to satisfy constraints or any $r_{i,j} \geq b_{ij}$, go to step 11.

Step 9. Search for another available tile $v \in V$.

Step 10. Reset the tile v assigned to this IP u and go to step 5.

Step 11. If the unmapped stack is not empty, go to step 4.

Fig. 6. Mapping algorithm.

$u_i \in$ CTG(U, E) has been mapped onto the NAG(V, L) yet. Each internal node stands for a partial mapping of $u_i \in$ CTG(U, E) onto $v_i \in$ NAG(V, L). For example, the leftmost node in level 2 represents a partial mapping, where u_0 and u_1 are mapped onto v_0 and v_1, respectively, while the u_2 and u_3 are still unmapped. If a node does not meet the constraints, this node cannot branch into subnodes. Every leaf node stands for a complete mapping satisfying constraints. To keep the figure simple, we do not show all nodes, and the algorithm typically does not need to search all paths.

Our mapping algorithm is based on the branch-and-bound strategy. The selected bound affects the search result. In [7], the total communication bandwidth requirements between mapped IPs and the estimated communication bandwidth requirements between unmapped IPs are used to find a mapping with low-power consumption. In our case, the major problem is latency. We use n_{hops} to cut down the solution space. If n_{hops} of p_{ij} between mapped IPs is larger than the n_{hops} of this p_{ij} in constraints, we can conclude that this node will not lead to a feasible solution.

The mapping algorithm is shown in Fig. 6. First, the IPs whose communication latency is under latency constraint have high priority to be mapped. The remaining IPs are sorted according to the bandwidth requirements of the communication flows in decreasing order and are stored in an unmapped stack. The smallest is placed at the bottom of the stack. When an IP (top of stack) is selected to map onto a tile, all circuits related to this IP are established based on an XY routing algorithm. Meanwhile, $n_{\text{hops}_{i,j}}$ defined in (2) and all $r_{i,j}$ related to this IP are calculated.

Then, $n_{\text{hops}_{i,j}}$ is calculated to ensure this mapping satisfies latency constraints, and r_{ij} is calculated to ensure the bandwidth of links can satisfy the requirement of circuits. When an IP is mapped successfully, it is pushed into the mapped stack. If an IP cannot be mapped to any available tile, the algorithm pops and re-maps an IP from the mapped stack to obtain different mapping results. This algorithm repeats until all IPs in the CTG are mapped. The goal of this algorithm is to efficiently search for a satisfying mapping rather than find the optimal mapping due to its complexity. Mapping an IP requires the time to search for an available tile v in V, establish corresponding p, and calculate r_{ij}, n_{hops}.

2. Scheduling Algorithm

After mapping is completed, the scheduling algorithm assigns the time slots to the circuits established by the mapping algorithm. We divide the scheduling problem into the local problem for each switch and the global problem for the whole network. Our algorithm solves these problems separately.

The scheduler consists of three steps in the main algorithm in Fig. 3. Step 3 is for the local problem and it arranges the connections for each switch in the NoC. Step 4 is for the global problem and it assigns the time slots to the circuits. Since the output time of a switch corresponds to the input time of the neighboring switch, the time slot assignment solves the global problem. Step 5 tries to minimize the latency of the circuits.

Figure 7 shows an example of the mapping result. In Table 1, we present the connection requirement for each switch using the example of S4. The table shows the connection requirement of S4, assuming that there are 8 time slots in each frame. In the connection requirement table (CRT), each element indicates the circuit connection and the bandwidth requirement. For example,

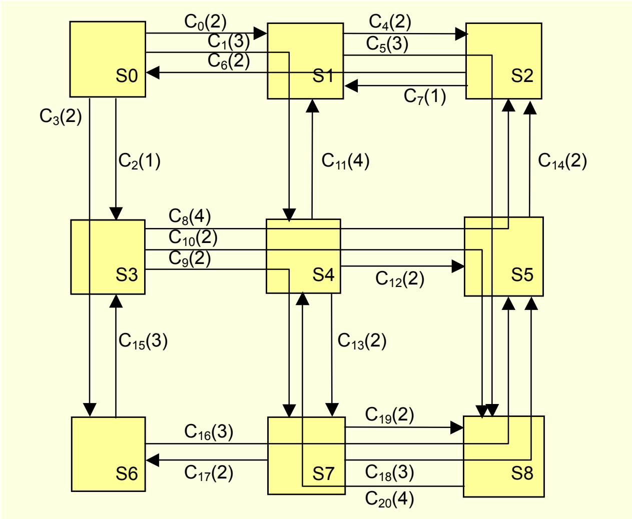

Fig. 7. Communication circuits in an NoC chip.

the element $C_{12}(2)$ indicates that circuit C_{12} enters the input In 0 and departs the output Out 1. The bandwidth requirement of such a connection is two time slots.

The basic objective of step 3 is to arrange the connections in a switch for finding a contention-free match based on the CRT. Consider an $N \times N$ switch consisting of N inputs and N outputs. A CRT can be represented by a traffic matrix, which is denoted as T. Let $M_k = (m_{ij})$ be the matching matrix for time slot k, which indicates the match between inputs and outputs at time slot k. If input i and output j match, then $m_{ij} = 1$. Otherwise, $m_{ij} = 0$. Such a matrix is a permutation matrix. The arrangement is admissible if and only if the following equality is satisfied:

$$\sum_{k=0}^{S-1} M_k = T, \tag{4}$$

where S is the number of time slots in a frame.

The arrangement problem can be reduced to the matrix decomposition problem. The Birkhoff-von Neumann decomposition theorem, which decomposes any $N \times N$ doubly stochastic matrix, provides the best upper bound N^2-$2N$+2 on the number of permutation matrices. A matrix $D = (d_{ij})$ is a doubly stochastic matrix [15] if it satisfies the following:

$$\sum_{i}^{N-1} d_{i,j} = 1, \ \forall j \quad \text{and} \quad \sum_{j}^{N-1} d_{i,j} = 1, \ \forall i .$$

In our problem, the number of permutation matrices cannot be greater than the number of time slots in a frame because the matrix of the CRT is a special case in which the sum of each row or column is not greater than the number of time slots. We can decompose such a matrix into P permutation matrices, where P is not greater than the number of time slots. Before describing our algorithm, we give some definitions related to

Table 1. Connection requirement table for S4.

	Out 0	Out 1	Out 2	Out 3	Out 4
In 0		$C_{12}(2)$	$C_{11}(4)$		$C_{13}(2)$
In 1					
In 2	$C_1(3)$				
In 3		$C_{10}(2)$, $C_8(4)$			$C_9(2)$
In 4	$C_{20}(4)$				

Decomposition (T, M)

Step 1. Set i=1.

Step 2. Select a minimum element ϕ_i in the matrix T.

Step 3. Use bipartite match to find a maximum matching G of the matrix T.

Step 4. Construct a permutation matrix $M[i]$ based on G.

Step 5. Deletion. Set $T = T - \phi_i\, M[i]$.

Step 6. If T has any nonzero element, set i=i+1 and go to step 2; otherwise, end.

Fig. 8. Decomposition routine.

our algorithm.

Define an $N \times N$ matrix $T = (t_{ij})$, where T is a request matrix if it satisfies

$$\sum_{i=0}^{N-1} t_{i,j} \le S, \ \forall j \tag{5}$$

and

$$\sum_{j=0}^{N-1} t_{i,j} \le S, \ \forall i , \tag{6}$$

where S is the number of time slots in a frame. A request matrix T can be expressed as a linear combination of permutation matrices:

$$T = \sum_k \phi_k P_k , \tag{7}$$

where P_k is a permutation matrix, ϕ_k is an integer, and $0 < \phi_k \le S$ such that $\sum_k \phi_k \le S$.

We use the bipartite matching algorithm to decompose the matrix. The decomposition routine, shown in Fig. 8, repeatedly performs maximum matching on the nonzero elements of the matrix. Such matching must involve the nonzero minimum element in this matrix. This ensures that at least one element of this matrix is zeroed per iteration. Then, the minimum element is subtracted from the selected elements in the matrix, and this procedure is repeated until all elements of the matrix have been zeroed. Figure 9 shows the decomposition of the matrix based on the CRT for S4 from Table 1. According to the bipartite match

$$\begin{bmatrix} 0&2&4&0&2\\ 0&0&0&0&0\\ 3&0&0&0&0\\ 0&6&0&0&2\\ 4&0&0&0&0 \end{bmatrix} = 2\times\begin{bmatrix} 0&1&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&1\\ 1&0&0&0&0 \end{bmatrix} + \begin{bmatrix} 0&0&4&0&2\\ 0&0&0&0&0\\ 3&0&0&0&0\\ 0&6&0&0&0\\ 2&0&0&0&0 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 0&0&4&0&2\\ 0&0&0&0&0\\ 3&0&0&0&0\\ 0&6&0&0&0\\ 2&0&0&0&0 \end{bmatrix} = 2\times\begin{bmatrix} 0&0&0&0&1\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&1&0&0&0\\ 1&0&0&0&0 \end{bmatrix} + \begin{bmatrix} 0&0&4&0&0\\ 0&0&0&0&0\\ 3&0&0&0&0\\ 0&4&0&0&0\\ 0&0&0&0&0 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 0&0&4&0&0\\ 0&0&0&0&0\\ 3&0&0&0&0\\ 0&4&0&0&0\\ 0&0&0&0&0 \end{bmatrix} = 3\times\begin{bmatrix} 0&0&1&0&0\\ 0&0&0&0&0\\ 1&0&0&0&0\\ 0&1&0&0&0\\ 0&0&0&0&0 \end{bmatrix} + \begin{bmatrix} 0&0&1&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&1&0&0&0\\ 0&0&0&0&0 \end{bmatrix}$$

(c)

Fig. 9. Decomposition of the matrix based on the CRT for S4 from Table 1.

```
#define MAX_NoC_SIZE    M    /*maximum NoC size*/
#define MAX_TIME_SLOT    S    /*number of time slots*/
typedef struct    output_port {
/* define the data structure of an output port */
            int I[S];
            int CN[S];
            int ITS[S];
            int WT[S];
            };
typedef struct OTST{
/*define the data structure of an output time slot table*/
            int OTS[S];
            int TWT[S];
            output_port Out[5];
            };
OTST O[M];
```

Fig. 10. Data structure of the OTST.

Table 2. Output time slot table of S4.

OTS	Out 0				Out 1				Out 2				Out 3				Out 4				TWT
	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	
					3	8			0	11											
	4	20			0	12											3	9			
	4	20			0	12											3	9			
	4	20			3	8											0	13			
	4	20			3	8											0	13			
	2	1			3	8			0	11											
	2	1			3	10			0	11											
	2	1			3	10			0	11											

including the minimum element, the selected elements are $e(0, 1)$, $e(3, 4)$, and $e(4, 0)$. Here, $e(x, y)$ represents the element in row x and column y. The value of the minimum element is 2. The values of the selected elements are decreased by 2. Figure 9(a) shows the result of the first cycle of the algorithm.

Given the bound on the number of time slots, S, the request matrix can be decomposed into m permutation matrices, where m is not greater than S. The time-complexities of steps 2 to 5 are N^2. The maximum number of cycles of the algorithm is S. Thus, the time-complexity of the decomposition routine is $O(N^2S)$.

The result of the decomposition is stored in the output time slot table (OTST) shown in Table 2. In Table 2, OTS represents the output time slot, and TWT represents the total waiting time. Each output in the OTST has four items, I, CN, ITS, and WT. These fields represent the input port, the circuit number, the input time slot, and the waiting time, respectively. In this step, only input ports and circuit numbers are stored into this table. The data structure of the OTST is shown in Fig. 10. Here, $O[j]$ represents OTST of switch Sj. Thus, assignment $O[4].Out[1].I[0]$=3 indicates that time slot 0 of output port 1 of switch S4 is assigned to the packet from input port 3.

After the connections of all switches are arranged, the time slots have yet to be assigned to all circuits. The assignment of the time slots is a global problem because the output time of a switch affects the input time of neighboring switches. In Fig. 7, switch S4 connects to four neighboring switches: S1, S3, S5, and S7. Circuits C_{11}, C_{12}, and C_{13} start from switch S4. If the start time of C_{11} is assigned, the input time of C_{11} entering S1 is decided.

Step 4 assigns the start time to all circuits in the NoC. Then, it assigns the input time of all circuits to the related OTSTs. In this step, the waiting time of the input data is also calculated. The waiting time is defined as the duration for a packet stored in the buffer. In our simulations, we assume that the switch architecture is unconstrained (zero overhead) and the latency of the link is zero because we only want to reduce the waiting time. The waiting time is defined as

$$T_{\text{waiting}} = (OTS - ITS + S) \bmod S\,. \qquad (8)$$

We use a routine, called time slot allocation (TSA) shown in Fig. 11, to assign the OTS for each row in each OTST. In each assignment, the routine must assign the ITS of the related circuits to the neighbors' OTSTs. It searches the CN fields in the OTSTs of four neighboring switches consecutively. In each search, having found the CN, this routine fills in the ITS with a proper time slot. At the same time, it calculates the WT and

```
Time_Slots_Allocation (OTSTs):
Step 1. Set i, s = 0.
Step 2. Assign O[i].OTS[s]=s;
Step 3. Input time assignment and waiting time calculation.
        For j = 0 to N-1
        If O[i].Out[j].IN[s]=0 then   /* local input port */
            O[i].Out[j].TS[s]=s;
            O[i].Out[j].WT[s]=0;
        else
            Use O[i].Out[j].CN[s] as key to search the CN fields in
            the OTST of the switch which input port connects to
            O[i].Out[j]. Assign corresponding TS and calculate its
            WT. Update corresponding TWT.
Step 4. If s < S then s=s+1 and go to step 3.
Step 5. If i < M, i=i+1 and go to step 2. Otherwise end.
```

Fig. 11. Time slot allocation routine.

Table 3. Time slot assignment of the OTST of S4.

OTS	Out 0				Out 1				Out 2				Out 3				Out 4				TWT
	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	
0					3	8			0	11	0	0									
1	4	20			0	12	1	0									3	9			
2	4	20			0	12	2	0									3	9			
3	4	20			3	8											0	13	3	0	
4	4	20			3	8											0	13	4	0	
5	2	1			3	8			0	11	5	0									
6	2	1			3	10			0	11	6	0									
7	2	1			3	10			0	11	7	0									

Table 4. OTST of S5.

OTS	Out 0				Out 1				Out 2				Out 3				Out 4				TWT
	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	I	CN	ITS	WT	
0	4	16							0	14							3	10			
1	4	16							0	14							3	10			
2	4	16							3	8	0	2					2	5			2
3	4	18							3	8	3	0					2	5			0
4	4	18							3	8	4	0					2	5			0
5	4	18							3	8	5	0									0
6	3	12	1	5																	5
7	3	12	2	5																	5

updates the TWT. Because we assume the latency of both the switch and the link is 0, the OTSTs for a real NoC system must be adjusted according to the latencies of the switch and the link in this NoC. We only need to rotate the I fields based on the latency of the switch and add the ITS fields according to the latency of the link.

We use an example to illustrate the operation of this algorithm. The OTSTs of switches S4 and S5 are shown in Tables 3 and 4, respectively. In Table 3, the TSA algorithm assigns the value of 0 to the first OTS. Because Out 1 of S4 connects to In 3 of S5, this algorithm uses $O[4].Out[1].CN[0]=8$ (which indicates C_8) to search the OTST of S5, $O[5]$. Circuit C_8 is found in $O[5].Out[2].CN[2]$. TSA fills in the field $O(5).Out[2].TS[2]$ with 0 and calculates the waiting time, $O(5).Out[2].WT[2]=2\text{-}0=2$. The operation is repeated until all OTSTs have been assigned. Searching a circuit number in an OTST costs $N\times S$. Thus, the time-complexity of Step 3 is N^2S. Fulfilling an OTST, step 3 is executed S times. The time-complexity of the TSA routine is $O(N^2S^2M)$.

To lower the latency of the switch, the waiting time should be reduced. The waiting time of a switch depends on when the input data is delivered by the output. We can swap the rows in the OTST to reduce the total waiting time. Swapping rows in the OTST of a switch changes the output times of circuits in the swapped rows, and the related input time slots in the OTSTs of four neighboring switches need to be reassigned. Furthermore, decreasing the TWT for an OTST may increase the TWTs of other OTSTs; therefore, in the algorithm, we need a global variable that records the total TWTs to lower the average latency of the NoC.

The latency minimization (LM) algorithm is used to implement step 5 in Fig. 3. It minimizes the total waiting time by swapping rows in the OTST. In this routine, the Local_Waiting variable keeps the total waiting time in the swapped OTST. The Global_Waiting variable keeps the total waiting time of all OTSTs. First, this algorithm uses Local_Waiting as a key to construct a global max heap. The OTST corresponding to the root of the heap is selected to minimize its waiting times.

When an OTST is to be adjusted, we use the TWTs as key values to create a local max heap LH. The root of the max heap needs to be swapped first. In each swapping step, the maximal value of the WT fields in the row of the root is used to decide which row is selected to be swapped first. For instance, if the maximum value of WT is 5 in row 7, we can swap row 7 with row 2 to eliminate the maximum value of WT in row 7. If this swap can lower both Local_Waiting and Global_Waiting, it is done. After two rows have been swapped, the related fields in the neighbor's OTSTs are recalculated. Local_Waiting and Global_Waiting are updated, and LH is adjusted.

If this swapping cannot decrease both Local_Waiting and Global_Waiting, the algorithm selects the next rows to swap.

When all other rows in this OTST have been selected, the root of LH is deleted. The swapping operation repeats until LH is empty.

IV. Experimental Results

We used an open tool, Task Graph For Free (TGFF) [16], to generate the test pattern for simulating and evaluating our algorithm. We also developed a GUI tool for users to employ to evaluate their NoC designs. This tool provides several mechanisms to help design the crucial circuit and to help users to set the design constraints.

Our simulation platform was a microcomputer. The CPU is an Intel® Core™ 2 Quad @2.4GHz processor, its main memory is 2 GB, and its operating system is Windows XP professional. Because of the limited length of this paper, we only present some of the simulation results. Figure 12 compares the average waiting times of the circuit in a switch for three schemes: the TSA only (Fig. 11, no LM step), LM, and the optimal mapping and scheduling with 6×6, 8×8, and 10×10 mesh NoC architectures with 8 time slots per frame. The TSA (limited) denotes the result by using the TSA scheduler with latency constraints. We randomly generate the constraints for circuits and their latencies. The average number of circuits of 5 tasks with 10×10 mesh NoC architectures is 149.8. After mapping and scheduling by TSA, TSA (limited), LM, LM (limited), optimal, and optimal (limited), the average waiting times are 1.2, 1.7, 0.61, 0.86, 0.34, and 0.52, respectively. Our results show that the LM algorithm achieves a significantly better average waiting time than the TSA algorithm, and it is close to that of the optimal scheduling.

We can use the branch-and-bound algorithm to get an optimal solution. The result of LM can be used as the bound to find an optimal scheduling. Even if we use the branch-and-bound algorithm, the runtime required for optimal scheduling is half an hour or more. The numbers of tiles, time slots, and circuits affect the runtime of the algorithm. For 10×10 mesh NoC architectures, the runtimes of LM and LM (limited) simulations need tens of minutes. However, the runtime of optimal scheduling needs dozens of hours with the branch-and-bound algorithm.

Different mappings lead to different scheduling results. We compare the mapping algorithm in [8] with ours. After mapping, both mapping algorithms use our scheduling algorithm to schedule the time slot. Figure 13 shows the results. The average hops of circuits in the 10×10 mesh NoC architecture of our mapping and the mapping in [8] are 3.73 and 3.72, respectively. The average waiting times of LM using our mapping and the mapping in [8] are 0.611 and 0.614, respectively. Therefore, our mapping is comparable to that of [8] and we have a good scheduling algorithm to complete the task.

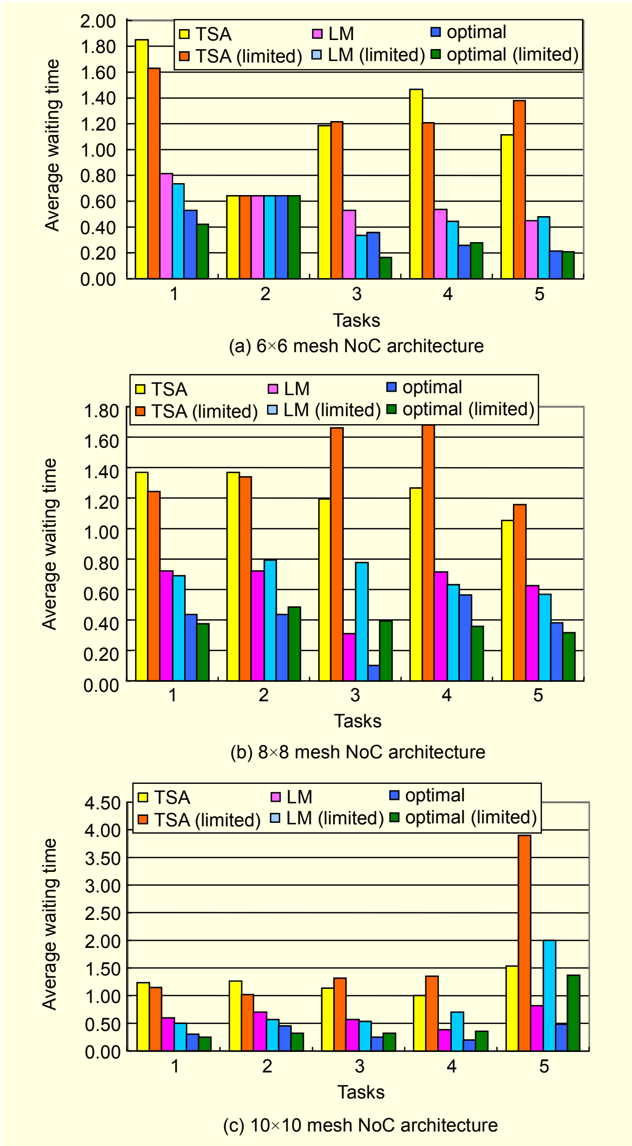

Fig. 12. Comparisons between the TSA, LM, and optimal algorithms.

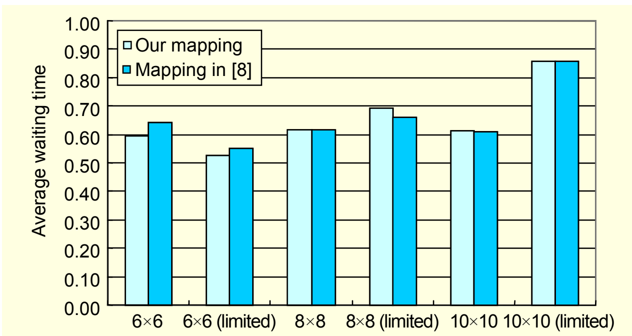

Fig. 13. Results of LM compared to the mapping algorithm in [8].

The optimal mapping takes too much time; therefore, we do not want to use an exhaustive search to find the optimal mapping in our design due to its complexity. Simulation results show that LM performs well and the result is not far from an optimal solution. In some cases, the average waiting times of the results with latency constraints are smaller than those of the scheduler with no latency constraints. Overall, the results reflect that the LM algorithm is a good and feasible method for the mapping and scheduling in a circuit-switched NoC architecture.

V. Summary and Conclusion

We proposed an efficient mapping and scheduling scheme for the NoC design. The algorithm is suitable for circuit-switched networks or those that need to preserve the communication bandwidth for guaranteed services. We also developed a tool to set the design constraints for users. The simulation results demonstrate that our LM scheme provides an effective solution for the circuit-switched NoC design. However, future SoC chips may contain thousands of IPs. More simulation work needs to be carried out for large-size networks for future large-scale SoC. In addition, we plan to apply our scheduler to real SoC design applications in order to gather more results and adjust our scheduler design.

References

[1] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, Jan. 2002, pp. 70-78.

[2] W. J. Dally and B. Towels, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proceedings of 38th Design Automation Conference*, Las Vegas, Nevada, USA, June 2001, pp. 684-689.

[3] K. Goossens, J. Dielissen, and A. Rădulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Trans. on Design and Test of Computers*, vol. 22, no. 5, Oct. 2005, pp. 414-451.

[4] S. Kumar et al., "A Network on Chip Architecture and Design Methodology," *Proc. of the IEEE Computer Society Annual Symp. on VLSI, 2002 (ISVLSI.02)*, Pittsburgh, PA, USA, Apr. 2002, pp. 105-112.

[5] A. Hemani et al., "Network on Chip: An Architecture for Billion Transistor Era," *Proc. of IEEE NorChip Conference*, Turku, Finland, Nov. 2000, pp. 166-173.

[6] D. Wingard, "MicroNetwork-Based Integration for SOCs," *Proceedings of 38th Design Automation Conference*, Las Vegas, Nevada, USA, June 2001, pp. 673-677.

[7] J. Hu and R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures," *Proc. of Design Automation and Test in Europe Conference and Exhibition*, Munich, Germany, Mar. 2003, pp. 688-693.

[8] S. Murali and G.D. Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," *Proc. of Design Automation and Test in Europe Conference and Exhibition*, Paris, France, vol. 2, Feb. 2004, pp. 896-901.

[9] T. Lei and S. Kumar, "A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," *Proc. of Euromicro Symp. on Digital System Design: Architectures, Methods, and Tools*, Belek-Antalya, Turkey, Sept. 2003, pp. 180-187.

[10] D. Wiklund and D. Liu, "SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems," *Proc. of Int'l Parallel and Distributed Processing Symposium*, Nice, France, Apr. 2003, p.78.

[11] Hsin-Chou Chi and Chia-Ming Wu, "Efficient Switches for Network-on-Chip Based Embedded Systems," *Proc. of IFIP Int'l Conf. on Embedded and Ubiquitous Computing*, Nagasaki, Japan, vol. 3824, Dec. 2005, pp. 67-76.

[12] E. Rijpkema et al., "Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip," *Proc. of Design, Automation and Test in Europe*, Munich, Germany, vol. 150, no. 5, Sept. 2003, pp. 294-302.

[13] B. Towles and W.J. Dally, "Guaranteed Scheduling for Switches With Configuration Overhead," *IEEE/ACM Trans. on Networking*, vol. 11, no. 5, Oct. 2003, pp. 835-847.

[14] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. on Computers*, vol. C-36, no. 5, May 1987, pp. 547-553.

[15] G.D. Birkhoff, "Tres Observaciones Sobre el Algebra Lineal," *Universidad Nacional de Tucuman Revista*, Series A, vol. 5, 1946, pp. 147-151.

[16] R.P. Dick, D.L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," *Proc. of the 6th Int'l Workshop on Hardware/Software Co-design*, Seattle, Washington, USA, Mar. 1998, pp. 97-101.

Chia-Ming Wu received the BS degree in electronic engineering from National Taiwan University of Science and Technology in 1992, and the MS degree in computer science and information engineering from National Dong Hwa University, Hualien, Taiwan, in 2000. He is currently working toward the PhD degree in computer science and information engineering at National Dong Hwa University. His research interests include interconnection networks, network algorithms, and VLSI design.

Hsin-Chou Chi received the BS and MS degrees in electrical engineering from National Taiwan University, and the PhD degree in computer science from UCLA. He is currently an associate professor of the Department of Computer Science and Information Engineering and Director of the Computer and Information Technology Center at National Dong Hwa University, Hualien, Taiwan. His research interests include computer architecture, VLSI design, and embedded systems.

Ruay-Shiung Chang received his BSEE degree from National Taiwan University in 1980, and his PhD degree in computer science from National Tsing Hua University in 1988. He is now a professor in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. His research interests include Internet, wireless networks, and grid computing. Dr. Chang is a member of ACM, a senior member of IEEE, and a founding member of ROC Institute of Information and Computing Machinery. Dr. Chang also served on the advisory council for the Public Interest Registry (www.pir.org) from 2004/5 to 2007/4.