A Throughput-Enhanced Parallel Scheduling Algorithm for the MIQ Switch with a Moderate Number of Queues

Hakyong Kim, Kiseon Kim, Hyunho Yoon, and Yongtak Lee

Indexing terms: MIQ switch, VOQ switch, scheduling algorithm

A novel parallel scheduling algorithm, namely a parallel solitary-request-first (*PSRF*) algorithm, is proposed to improve the throughput performance of the MIQ switch when the number of queues are less than the switch size.

Introduction: In order to overcome the throughput limit of the single input-queued (SIQ) packet switch, a number of buffering strategies have been put forward in recent years. Among them, the virtual output-queued (VOQ) switch is receiving considerable attention, since it can yield 100% throughput depending on the scheduling algorithm employed. As a special case of the multiple input-queued (MIQ) switch which uses an idea of deploying multiple queues in each input, the VOQ switch equips each input with the same number of queues as the switch size and each queue stores the cells for the same destination. It implies that the VOQ switch eliminates the HOL blocking observed in the SIQ switch and, as the result, leads to the improved performance.

Shortcomings of the VOQ switch are (1) that the switch requires N^2 queues when the switch size is N, and (2) that the controller has to manage those queues simultaneously. These cost the implementation of the VOQ switch prohibitively and make the switch infeasible, especially for large dimensions. In addition, the performance enhancement by increasing the number of queues in the MIQ switch is not significant when the number of queues is larger than 4 [1, 2]. In reference [1],

specially, the author found that the hardware implementation and the performance of the MIQ switch is most reasonable when the number of queues is 2 or 4. In this Letter, therefore, we introduce a parallel scheduling algorithm which improves the switch performance for moderate number of queues.

PSRF scheduling algorithm: The parallel solitary-request-first (PSRF) algorithm introduced in this Letter is a parallel algorithm based on a sequential algorithm, called Chessboard [3], and the three-phase scheme, represented by PIM [4], which is consisted of Request, Grant, and Accept phases.

Before describing the proposed algorithm, let's define the solitary request be a request existing uniquely in a specific output in Request phase, the solitary output be an output receiving the solitary request, and the solitary input be an input receiving a grant from the solitary output. In Grant phase of the three-phase scheme, then, the solitary output always grants the corresponding input (solitary input). The grant from the solitary output, however, may or may not be selected by the input in Accept phase when the input receives other grants as well. It renders the solitary output idle during the time slot, resulting in a poor performance. To relieve such solitary requests which were lost in Accept phase, conventional schemes, such as PIM, iterate their three phases several times.

In the PSRF algorithm, which is described in Fig. 1, the scheduler accepts the grant from the solitary output with preference. The three phases of the PSRF algorithm are as follows:

Request phase Each input sends a request to every output for which it has a queued cell.

Grant phase When an output receives the solitary request, it chooses the request and marks the solitary input and solitary output. When an output receives two or more requests, it chooses one randomly among the requests from the non-

solitary input. The inputs corresponding to the selected requests are granted.

Accept phase When an input receives only one grant, it accepts the grant. When an input receives more grants than one, it chooses one randomly among the grants from the solitary outputs. If all grants are from the non-solitary outputs, the input accepts one by randomly selecting an output. The outputs corresponding to the selected grants are notified.

The idea behind PSRF, similar to that for Chessboard, is well well explained via a simple example of Fig. 2, based on a 2×2 MIQ switch where each input has 2 separate queues. In input 1, there is only one cell for output 2 (r_{12}), while, in input 2, there are two cells, each for output 1 (r_{21}) and output 2 (r_{22}), respectively. Request r_{21}, output 1, and input 2 correspond to the solitary request, solitary output, and solitary input, respectively. In Grant phase, therefore, the PSRF scheduler will select the solitary request r_{21} first and the request r_{12} from the non-solitary input. From this example, it is manifest that the selection of r_{12} and r_{21} gives us the highest performance.

Fig. 3 plots the curves of the saturation throughputs versus number of queues (m) in an input port, which is obtained for the i.i.d. Bernoulli traffic by computer simulation. The solid line and the dashed line represent the result for PSRF and for PIM, respectively, when we iterate PIM once. N designates the switch size and m ranges from 1 to N. As shown in the figure, PSRF outperforms the PIM algorithm. When m is equal to 2, 4, or 8, the saturation throughput of PSRF is higher than that of PIM by 5 to 6% and higher than that of the SIQ switch by 8 to 10%. Notice that the throughput for PSRF is maximum when m is 4 except for the case that N is equal to 8, while the throughput for PIM is increasing with m. It is also noticeable that the results for PSRF and PIM are equal to each other when m is equal to N, i.e., the VOQ switch. This stems from the fact that, as the number of queues increases, the probability that an output receives the solitary request becomes smaller

and, as the result, PSRF becomes indiscernible from PIM.

Results and discussion: In this Letter, we have proposed a parallel scheduling algorithm for the MIQ switch. The proposed PSRF algorithm improves the throughput performance of MIQ switches with a moderate number of queues by selecting the solitary request with preference. Computer simulation results show that the throughput of PSRF is enhanced when the number of queues are 2 or 4. It implies that switches, using the PSRF algorithm as well as a moderate number of queues, needs lower hardware complexity and control time, which result contributes to a high-performance and high-speed switch.

References

1. THOMAS, G.: 'Bifurcated queueing for throughput enhancement in input-queued switches', *IEEE Commun., Letters*, March 1997, 1 (2), pp. 56-57.

2. KIM, H., OH, C., LEE. Y., and KIM, K.: 'Throughput analysis of the bifurcated input-queued ATM switch', *IEICE Tr. Communi*, May 1999, E82-B (5), pp.768-772.

3. KIM, H., KIM, K, LEE, Y., YOON, H., and OH, C.: 'A simple and efficient cell selection algorithm for the multiple input-queued ATM switch', *Proc. IEEE ATM Workshop'99*, Kochi, Japan, 24-27 May 1999, pp.259-264.

4. ANDERSON, T.E., OWICKI, S.S., SAXE, J.B., and THACKER, C.P.: 'High-speed switch scheduling for local-area networks', *ACM Tr. Computer Systems*, Nov. 1993, 11 (4), pp.319-352.

Authors' affiliations:

Hakyong Kim, Kiseon Kim and Yongtak Lee (Department of Information and Communications, Kwang-Ju Institute of Science and Technology (K-JIST), 1 Oryong-dong, Puk-gu, Kwang-Ju, 500-712, S. Korea)

e-mail: hykim@ieee.org or hykim@charly.kjist.ac.kr

Figure captions:

Fig. 1 Pseudo-code for the PSRF algorithm

Fig. 2 An example of the 2×2 MIQ switch where each input has 2 separate queues

Fig. 3 Saturation throughput of PSRF and PIM for a single iteration

——— PSRF

- - - - - PIM

Figure 1

```
/* i:input port number, j:output port number */
for (all i) Request(i) { /* Request phase */
/* Request to every output for which it has a queued cell */
}

for (all j) { /* Grant phase */
    if(num_request[j] == 1) { /* if output j is the solitary output */
        solitary_input[i] = YES;
        solitary_output[j] = YES;
}}

for (all j) Grant(j)
    if(solitary_output[j] == YES)
        Grant_request(i,j);     /* grant the solitary request */
    else if(solitary_output[j] == NO && num_request[j]>1) {
        i = Select(j);     /* select one request for output j */
        if(solitary_input[i] == NO) Grant_request(i,j);
}}

for (all i) Accept(i) { /* Accept phase */
    if(grant[i] == 1)
        Accept_grant(i,j);    /* accept the solitary grant */
    else if(grant[i] > 1) {
        j = Select(i);     /* select one grant for input i */
        if(solitary_output[j] == NO) Accept_grant(i,j);
}}
```

Figure 2

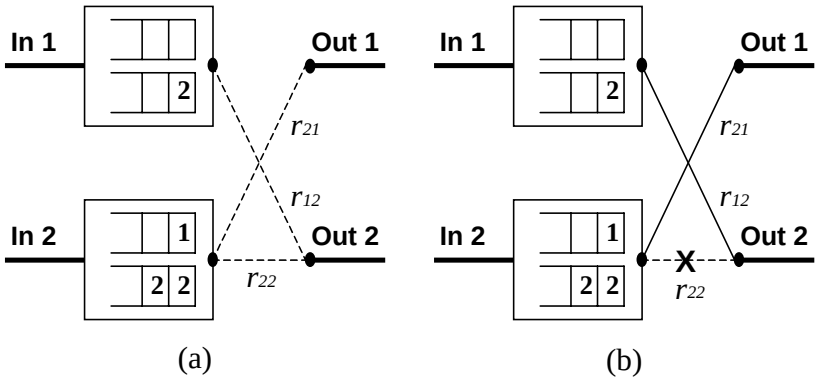

Figure 3