RZ 3636 (# 99646) 11/07/2005
Computer Science 3 pages

Research Report

Advanced Pipelined Crossbar Arbitration

Cyriel Minkenberg, Ilias Iliadis, François Abel

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Advanced Pipelined Crossbar Arbitration

Cyriel Minkenberg, Ilias Iliadis, François Abel

***Abstract*— Pipelined crossbar arbitration schemes overcome the time constraints of iterative matching algorithms, allowing support for higher line rates. One such scheme is the *fast low-latency parallel pipelined crossbar arbitration scheme* (FLPPR) introduced in [1]. We optimize its performance by applying advanced pre- and post-filter functions. By means of simulation we show that we can simultaneously achieve excellent latency–throughput characteristics under uniform traffic, high maximum throughput under nonuniform traffic, and prevent starvation.**

***Index Terms*— Packet switching, scheduling, pipelining.**

I. Introduction

Heuristic, parallel, iterative matching algorithms for input-queued cell switches with virtual output queuing (VOQ) require $O(\log N)$ iterations to achieve good performance. If the hardware implementation of the number of iterations required is not feasible within the cell duration, we can pipeline or parallelize the matching process to obtain a matching in every cell time slot. Examples of such schemes can be found in [1]–[6]. Most of the existing approaches are based on the same principle: As the matching process comprises a number S of serialized steps, the execution of these steps can be spread over multiple, say K, time slots. As a result, only S/K serialized steps need to be performed per time slot, thus relaxing the timing constraints. By computing K independent matchings in parallel in a time-shifted fashion, such that exactly one matching completes in a given time slot, the overall matching rate can be matched to the line rate.

We have proposed the *fast low-latency parallel pipelined arbitration* (FLPPR) framework in [1], which is tailored to low-latency applications such as interconnection networks for high-performance computing [7]. The most important difference to the related PMM scheme [5] is that, in every time slot, FLPPR allows requests to be submitted to all or any subset of the allocators (i.e., subschedulers) in parallel, instead of to just one specific allocator. This has some key advantages: First, latency at low utilization is independent of the number of pipeline stages, because there is a shortcut through the last allocator, thus allowing new requests to be granted immediately, without incurring the latency of the entire pipeline. In addition, this allows us to take into account the most recent arrivals in any matching, whereas in PMM the outcome of any matching is based only on arrivals up to K time slots ago. Finally, it achieves preferential treatment of long VOQs, which significantly improves performance under nonuniform traffic.

Figure 1 shows the FLPPR arbiter architecture, which comprises K pipeline stages. New requests are stored in the VOQ state unit. Depending on this state information (and

This research is supported in part by the University of California under subcontract number B527064.

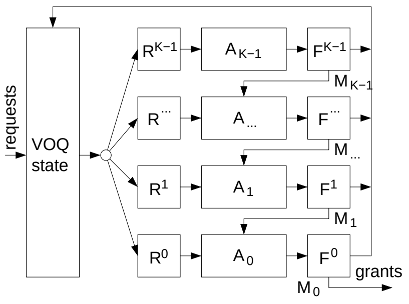


Fig. 1. FLPPR with allocators (A_k), pre- (R^k_{ij}) and post-filters (F^k_{ij}).

TABLE I
Methods 1, 2 & 3.

filter	Method 1	Method 2	Method 3
R^k_{ij}	$L_{ij} > 0$	$L_{ij} > 0$	$L_{ij} > k$
F^k_{ij}	$G_{ij} \leq L_{ij} \vee k = 0$	true	true

possibly other variables), the pre-filters R determine a set of requests per allocator. Every allocator computes a matching on the requests it receives, ignoring requests for ports already matched. The post-filters F may modify a matching before returning the results to update the VOQ state. Before the next time slot, every stage sends its current matching to the next stage. The matching from A_0 determines the grants to issue, whereas the matching of A_{K-1} is reset.

The performance characteristics depend on the *pre- and post-filter* functions that manipulate the requests and grants, respectively, based on variables such as VOQ length, new or existing matchings, or allocator position. L_{ij} denotes the length of VOQ Q_{ij}, G_{ij} denotes the number of new grants for Q_{ij}, m^k_{ij} denotes the current matching of A_k, n^k_{ij} denotes the newly matched edges of A_k, R^k_{ij} and F^k_{ij} denote the pre- and post-filters of Q_{ij} at A_k. Table I summarizes the pre- and post-filters for the existing methods 1, 2, and 3 introduced in [1]. Although these methods each exhibit some advantages, none achieves optimal performance in terms of mean latency under uniform traffic *and* maximum throughput under nonuniform traffic. Here, we will introduce new methods that use more sophisticated request and filter functions to optimize performance.

II. Advanced FLPPR

We improve the performance of the methods introduced in [1] by using more sophisticated pre- and post-filters. We first explore a method that employs an exact post-filter in Sec. II-A and one with an exact pre-filter in Sec. II-B. We then introduce a method that optimizes throughput with nonuniform traffic in Sec. II-C, and finally a method that in addition guarantees fairness in Sec. II-D.

A. Method 4: Broadcast requests, exact post-filtering

This method employs a broadcast request function as do methods 1 and 2, which can lead to excess grants. When there are excess grants, i.e., $G_{ij} > L_{ij}$, we cancel exactly $G_{ij} - L_{ij}$ grants, so that $G^{\star}_{ij} = L_{ij}$. To achieve low latency, we cancel the grants from the beginning of the pipeline, i.e., as close to A_{K-1} as possible. We first introduce the variable $k^{\star}_{ij}$. Assume $G_{ij} > L_{ij} \geq 1$. Consider the following function:[1]

$$g_{ij}(x) = \sum_{k=0}^{x} \mathbf{1}(n^k_{ij}),\ 0 \leq x < K. \tag{1}$$

Note that g_{ij} is a discrete, monotonically increasing continuous function on the interval $0 \leq x < K$, that $g_{ij}(0) \leq 1 \leq L_{ij}$, and that $L_{ij} < g_{ij}(K-1) = G_{ij}$. It follows that $\exists k^{\star}_{ij} \in [0, K-1]$, for which $g_{ij}(k^{\star}_{ij}) = L_{ij}$. If $G_{ij} \leq L_{ij}$, we define $k^{\star}_{ij} = K-1$. Note that, in general, $k^{\star}_{ij}$ is not uniquely defined. However, the selection of $k^{\star}_{ij}$ in such an ambiguous case does not affect the filtering outcome. Using $k^{\star}_{ij}$ as defined above, we define method 4 as follows:

$$R^k_{ij} \triangleq (L_{ij} > 0), \tag{2}$$
$$F^k_{ij} \triangleq (k \leq k^{\star}_{ij}). \tag{3}$$

Note that (2) corresponds to broadcasting requests and (3) to filtering out exactly the number of excess grants starting from the start of the pipeline. Note that (3) depends on n^k_{ij} (the set of newly matched edges) and k.

B. Method 5: Exact request function

This method applies pre-filtering to prevent excess grants from being issued by submitting no more than L_{ij} requests in parallel. The difference to method 3 is that method 5 takes into account which allocators have already been matched for the corresponding VOQ and skips those. Similar to the function g_{ij} introduced in Sec. II-A, we introduce h_{ij}:

$$h_{ij}(x) = \sum_{k=0}^{x} \mathbf{1}(\neg m^k_{ij}),\ 0 \leq x < K. \tag{4}$$

Analogous to the discussion on g_{ij}, if $G_{ij} > L_{ij}$ there exists a $k^{\star}_{ij} \in [0, K-1]$ such that $h_{ij}(k^{\star}_{ij}) = L_{ij}$. Otherwise, let $k^{\star}_{ij} = K-1$. Note that (4) counts the number of allocators in which an edge (i, j) has *not yet* been matched. Hence, submitting requests to all allocators A_k with $k \leq k^{\star}_{ij}$ will not result in any excess grants. Using $k^{\star}_{ij}$ as defined above, we define method 5 as follows:

$$R^k_{ij} \triangleq (L_{ij} > 0 \wedge k \leq k^{\star}_{ij}), \tag{5}$$
$$F^k_{ij} \triangleq \text{true}. \tag{6}$$

Note that (5) depends on m^k_{ij} (the set of existing edges) and k, and that no post-filtering is required.

[1] $\mathbf{1}(\cdot)$ is the identity function: $\mathbf{1}(\text{true}) = 1, \mathbf{1}(\text{false}) = 0$.

C. Method 6: Exact requests with threshold

Method 3 achieves better throughput under nonuniform traffic than methods 4 and 5 do because it reserves access to the first allocator A_{K-1} in the pipeline for long queues. Because long queues do not compete with short ones in this allocator, and because edges, once matched, will never be removed further down the pipeline, long queues receive preferential treatment, which achieves the same effect as a weighting scheme. However, because method 3 employs a simple pre-filter, it has poor performance under uniform traffic.

In method 6 we combine the pre-filter of method 5, thus obviating the need for post-filtering, with the preferential service for long queues of method 3. To achieve this, we introduce a threshold T, with $0 \leq T < K$. We define method 6 as follows, with $k^{\star}_{ij}$ as defined in Sec. II-B:

$$R^k_{ij} \triangleq (L_{ij} > 0 \wedge k \leq k^{\star}_{ij} \wedge (k < T \vee L_{ij} > T)), \tag{7}$$
$$F^k_{ij} \triangleq \text{true}. \tag{8}$$

The practical meaning of the threshold T is that all nonempty VOQs are allowed to submit requests to A_0 through A_{T-1}, whereas only VOQs with more than T packets are allowed to submit requests to A_T through A_{K-1}. Note that method 6 is identical to method 5 if $T = 0$ or $T = K$.

However, method 6 has a serious fairness issue: Short queues are never allowed to submit requests to A_{K-1}; so in principle they could be starved indefinitely.

D. Method 7: Exact requests with threshold and age

This method addresses the starvation issue of method 6 by adding an age a_{ij} to every VOQ. Age a_{ij} is reset to zero every time a packet from Q_{ij} is served. It is incremented in every time slot in which Q_{ij} is not served.

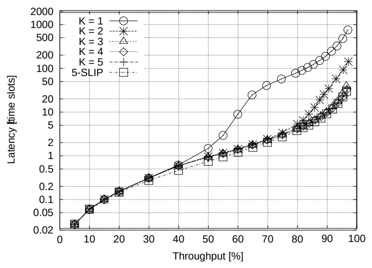
We employ an age threshold $A_{\max}$ to determine whether a short queue ($L_{ij} \leq T$) is eligible to submit a request to A_{K-1}. As a result, short queues can no longer be starved indefinitely, yet the weighting effect is preserved. We define method 7 as follows:

$$R^k_{ij} \triangleq \begin{cases} L_{ij} > 0 \wedge k \leq k^{\star}_{ij} & \text{if } a_{ij} \leq A_{\max}, \\ \wedge (k < T \vee L_{ij} > T) & \\ L_{ij} > 0 \wedge k = K-1 & \text{if } a_{ij} > A_{\max}, \end{cases} \tag{9}$$
$$F^k_{ij} \triangleq \text{true}. \tag{10}$$

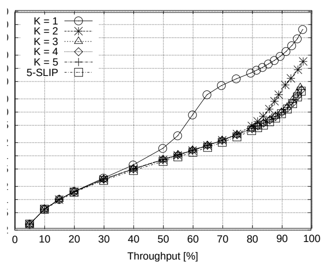
III. Performance

We study the performance of methods 4 through 7 by means of simulation. We use the steady-state simulation method to determine the mean throughput and latency with random traffic. Throughput is sampled at the switch egress in every time slot as the ratio of busy to total output ports. Latency is measured end to end and sampled for each packet delivered to the egress. The confidence intervals achieved are better than 0.2% with 99% confidence on the throughput, and better than 5% with 95% confidence on the latency. Note that the recorded minimum latency equals zero time slots (cut-through).

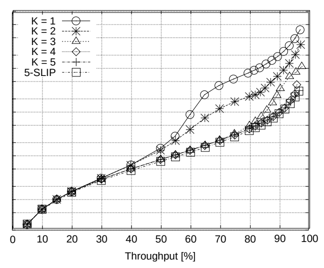
Figures 2(a–d) show the performance for $N = 32$ with uniform Bernoulli traffic for the proposed new FLPPR methods. Every allocator employs one iteration of the dual round-robin matching (DRRM) algorithm [8]. As a reference, the



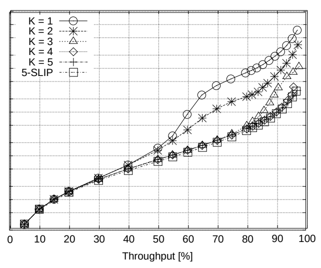
(a) Method 4



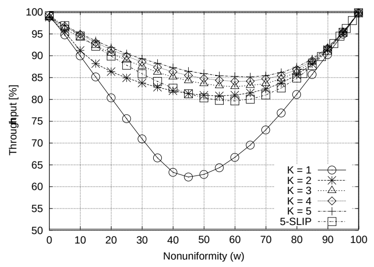
(b) Method 5



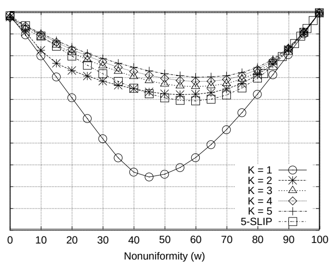
(c) Method 6



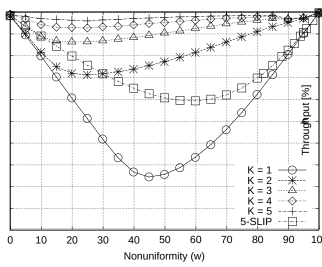
(d) Method 7



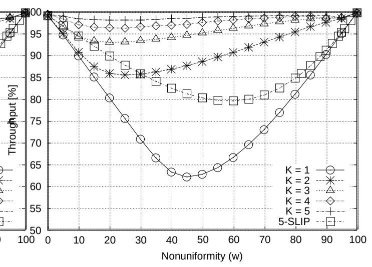
(a) Method 4



(b) Method 5



(c) Method 6



(d) Method 7

Fig. 2. (a) to (d): Latency–throughput with uniform Bernoulli traffic. (e) to (h): Throughput vs. nonuniformity w with Bernoulli traffic. $N = 32$.

performance of the i-SLIP algorithm [9] is also included, with $\log_2(32) = 5$ iterations *per time slot* (5-SLIP; K = 1). We let K range from 1 to 5, and let $T = K - 1$ for methods 6 and 7.

To study the performance under nonuniform traffic, we adopt a destination distribution characterized by a nonuniformity parameter w [10], where $w = 0$ corresponds to uniform traffic and $w = 1$ to fully unbalanced, contention-free traffic: $\lambda_{ij} = \lambda\left(w + \frac{1-w}{N}\right)$ if $i = j$, $\lambda\frac{1-w}{N}$ otherwise. Here, λ_{ij} represents the traffic intensity from input i to output j, $0 \le i, j < N$; λ is the aggregate offered load, and w the nonuniformity factor. Note that no input or output is oversubscribed and that traffic is admissible as long as $\lambda \le 1$. We vary the value of w from 0 to 1 and measure the throughput achieved at an offered load of 100%. Figures 2(e–h) show the results for $N = 32$ and Bernoulli traffic for K ranging from 1 to 5.

Figures 2(a–d) show that the latency–throughput characteristics under uniform traffic are excellent for all methods: There is a quick convergence on the reference 5-SLIP performance as K increases. All schemes perform better than any of the existing methods. Because methods 6 and 7 reserve A_0 for long queues, they need one allocator more to achieve this convergence. Figures 2(e–h) show that with methods 4 and 5 the maximum throughput with nonuniform traffic is limited to approx. 85% when $w = 0.6$. Methods 6 and 7, on the other hand, achieve close to 100% throughput under nonuniform traffic for any value of w with $K = 5$. Note that this is a considerable improvement, not only over methods 4 and 5, but also over the reference 5-SLIP.

It turns out the threshold setting $T = K - 1$ is optimal, which in effect reserves one allocator (A_0) for long queues. This is sufficient to achieve the preferential treatment; setting a lower T only reduces the effectiveness of the remaining allocators.

IV. Concluding Remarks

We proposed new pipelined crossbar arbitration schemes based on our FLPPR architecture by optimizing the pre- and post-filters to manipulate the requests and grants. We have shown that it is possible to simultaneously achieve excellent latency–throughput characteristics under uniform traffic, high maximum throughput under nonuniform traffic, and fairness. The added implementation complexity consists of adders and comparators, in particular to compute the value of $k^{\star}_{ij}$. Further FLPPR variants are possible; the framework could be extended to allow manipulation of existing edges (m^{k}_{ij}).

References

[1] C. Minkenberg, I. Iliadis, and F. Abel, “Low-latency pipelined crossbar arbitration,” in *Proc. IEEE GLOBECOM 2004*, Dallas, TX, Dec. 2004.

[2] P. Gupta and N. McKeown, “Designing and implementing a fast crossbar scheduler,” *IEEE Micro*, vol. 19, no. 1, pp. 20–28, Jan./Feb. 1999.

[3] A. Smiljanić, R. Fan, and G. Ramamurthy, “RRGS-round-robin greedy scheduling for electronic/optical terabit switches,” in *Proc. IEEE GLOBECOM 1999*, Rio de Janeiro, Brazil, Dec. 1999, pp. 584–555.

[4] D. Cavendish, “CORPS: A pipelined fair packet scheduler for high-speed input-queued switches,” in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2000*, Heidelberg, Germany, June 2000, pp. 55–64.

[5] E. Oki, R. Rojas-Cessa, and H. Chao, “A pipeline-based approach for maximal-sized matching scheduling in input-buffered switches,” *IEEE Commun. Lett.*, vol. 5, no. 6, pp. 263–265, June 2001.

[6] ——, “PMM: A pipelined maximal-sized matching scheduling approach for input-buffered switches,” in *Proc. IEEE GLOBECOM 2001*, vol. 1, San Antonio, TX, Nov. 2001, pp. 35–39.

[7] C. Minkenberg, F. Abel, P. Müller, R. Krishnamurthy, and M. Gusat, “Control path implementation of a low-latency optical HPC switch,” in *Proc. Hot Interconnects 13*, Stanford, CA, Aug. 17–19 2005, pp. 29–35.

[8] H. Chao and J. Park, “Centralized contention resolution schemes for a large-capacity optical ATM switch,” in *Proc. IEEE ATM Workshop*, Fairfax, VA, May 1998, pp. 11–16.

[9] N. McKeown, “The iSLIP scheduling algorithm for input-queued switches,” *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[10] R. Rojas-Cessa, E. Oki, and H. Chao, “CIXOB-k: combined input-crosspoint-output buffered packet switch,” in *Proc. IEEE GLOBECOM 2001*, vol. 4, San Antonio, TX, Nov. 2001, pp. 2654–2660.