

# Low-Latency Pipelined Crossbar Arbitration

Cyriel Minkenbergh, Ilias Iliadis, François Abel  
 IBM Research, Zurich Research Laboratory  
 Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland  
 {sil,ili,fab}@zurich.ibm.com

**Abstract**—Heuristic, parallel, iterative matching algorithms for input-queued cell switches with virtual output queuing 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, the matching process can be pipelined to obtain a matching in every cell time slot. However, existing approaches incur a substantial latency penalty because of the way the pipelining is performed. We introduce a new class of pipelined matching algorithms that can be based on any existing iterative matching algorithm, makes the minimum latency independent of the pipeline depth, and is highly amenable to distributed implementation. Our simulation results confirm that specific instances of this class achieve significantly lower average latency throughout the load range compared with existing schemes. We also propose an instantiation of the scheme that, in addition, significantly improves the performance with nonuniform traffic.

## I. INTRODUCTION

We consider crossbar-based input-queued (IQ) cell switches with virtual output queues (VOQs), as shown in Fig. 1. To

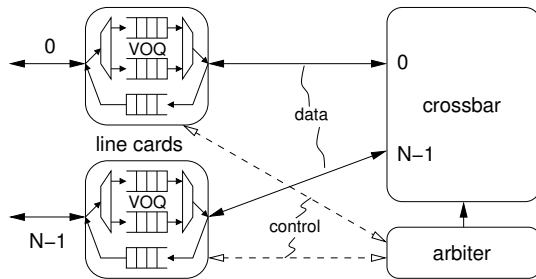


Fig. 1. System architecture, comprising  $N$  line cards containing the VOQs, an  $N \times N$  bufferless crossbar, and a centralized arbiter to resolve contention. We assume time-slotted operation with fixed-size packets.

obtain good performance, the centralized arbiter must compute a matching between inputs and outputs in every time slot. The known optimum solution to this problem [1] is too complex to be implemented in fast hardware. Instead, heuristic iterative algorithms such as PIM [2],  $i$ -SLIP [3], or DRRM [4], [5] are commonly used. The quality of their matching solution strongly depends on the number of iterations that can be carried out in the available arbitration time, i.e., in one time slot. In general,  $O(\log_2 N)$  iterations are required [2], [3] for adequate performance, although in the worst case these algorithms only converge in  $N$  iterations. As line

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

rates continue to increase but cell sizes remain constant, the arbitration time is shrinking, making it harder to complete enough iterations.

We denote the time to complete one matching iteration by  $T_i$ , the required number of iterations per arbitration by  $M$ , and the cell cycle by  $T_c$ .<sup>1</sup> The arbitration time  $T_M$  then is  $T_M = M * T_i$ . If  $T_M > T_c$ , pipelining can be used to obtain a matching in every time slot. One solution to this problem is to pipeline the matching process, as proposed in [6]–[12]. The pipelined maximal-sized matching (PMM) method proposed by Oki *et al.* [9], [10] is shown in Fig. 2. To obtain one arbitration decision at every time slot,  $K$  identical, parallel matching units (*allocators* [13] or *subschedulers* [9]) are employed, where  $K = \lceil T_M / T_c \rceil$ . However, the absolute minimum latency of this scheme, i.e., the cell latency in the absence of contention, is equal to  $K * T_c$  because an allocator must wait for all iterations to complete before returning a matching, even if this matching was produced in the first iteration. **In latency-sensitive applications, specifically in interconnects for parallel computers, such a latency penalty, which may be on the order of hundreds of nanoseconds, is highly undesirable. Our objective is to minimize the pipelining latency penalty for such applications, without compromising the maximum throughput.** Another related

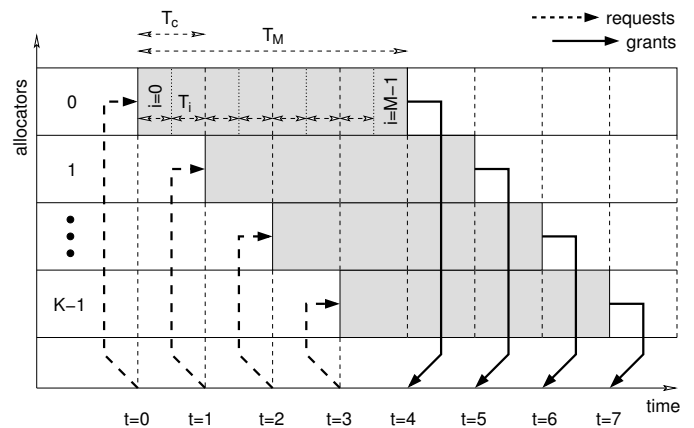


Fig. 2. PMM, according to [10].  $M = 8$ ,  $T_c = 2T_i$ , so  $K = \lceil T_M / T_c \rceil = 4$ .

scheme has been proposed in [11]: it comprises two *different*

<sup>1</sup>Without loss of generality, we assume that  $T_c$  is an integer multiple of  $T_i$ . The method presented here can also be applied if this is not the case. We use the terms cell cycle and time slot interchangeably.

allocators in a sequential setup: the first allocator executes one iteration of  $i$ -OCF and passes its resulting matching on to the second allocator, which may improve this matching by adding edges using the TSA (two-step arbiter) algorithm [14]. The objective is to reduce the number of  $i$ -OCF (iterative Oldest Cell First) iterations needed to one and simultaneously solve the starvation issue of TSA. However, the minimum latency in this scheme is equal to the sum of latencies of the two allocators.

The pipelined arbitration scheme (SPAA) used in the Alpha 21364 router [12] is based on exhaustive packet service. It allows incremental matching: an edge is only released for a new matching when all cells (flits) of the corresponding packet have been served, which may need several time slots. SPAA relies on the variable packet-length distribution in the Alpha 21364 system, which implies that, in a given time slot, arbitration is required for a small subset of the ports rather than for all ports in all time slots as is the case with fixed-size cells. Consequently, a simple single-iteration matching algorithm is sufficient in this context. In contrast, in the cases of non-exhaustive packet service or single-cell packets, this approach leads to very poor performance, even worse than single-iteration PIM, as shown in [12].

Burst switching schemes [15] can be used to address the same problem by enlarging the cell size, rather than decreasing the arbitration time. We do not consider this type of solution here, because it defeats our purpose of minimizing the latency.

We propose a scheme that simultaneously achieves

- a high maximum throughput owing to a pipelined implementation of the iterative arbitration process, and
- a minimum latency of just one time slot rather than  $K$  time slots as in existing schemes.

Most importantly, our scheme can be applied to pipeline the operation of any of the previously proposed, iterative bipartite-graph-matching algorithms.

## II. PARALLEL PIPELINED ARBITRATION

### A. Concept

The scheme proposed in [9] is actually more a parallelization than a pipeline because the different allocators exchange no information among each other. Our scheme also employs  $K$  allocators, but instead of a complete arbitration as in [9], every allocator executes only part of it and passes its result on to the next allocator in the *pipeline* as in [11]. Moreover, to reduce latency in the absence of contention, our scheme also provides the capability to send up to  $K$  requests in *parallel* and to collect the matching results produced after every pipeline stage. This concept, which we refer to as FLPPR (Fast Low-latency Parallel Pipelined aRbitration), is illustrated in Fig. 3. The gray boxes illustrate a specific sequence of  $M = 8$  iterations pipelined over  $K = 4$  allocators. Note that the requests are issued to all  $K$  allocators in parallel and that the first grants are issued at  $t = 1$  by  $A_0$  vs. at  $t = 4$  with PMM, see Fig. 2.

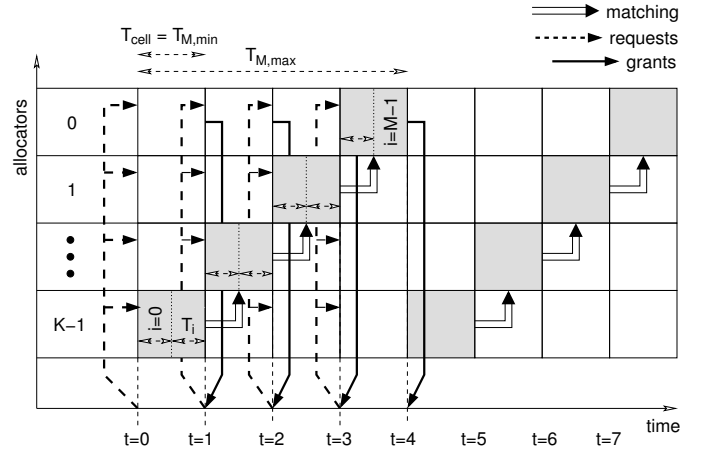


Fig. 3. FLPPR, with  $M = 8$  iterations and  $K = 4$  pipeline stages.

We now describe a new class of algorithms that is based on this arbiter architecture, and provide three instantiations of this class.

### B. Algorithm

We label the VOQs  $Q_{ij}$ ,  $0 \leq i, j < N$ . The central arbiter comprises  $N^2$  pending-request counters  $L_{ij}$ , corresponding to the number of unmatched cells of  $Q_{ij}$ . There are  $K$  pipeline stages with one allocator  $A_k$  per stage,  $0 \leq k < K$ . Every stage  $k$  maintains a matching  $\mathcal{M}_k = [m_{ij}^k]$ , where  $m_{ij}^k = \text{true}$  indicates that input  $i$  and output  $j$  have been matched to each other and  $m_{ij}^k = \text{false}$  otherwise. Initially,  $m_{ij}^k = \text{false}$  for all  $i, j, k$ . Depending on the specific matching algorithm  $\mathcal{A}$ , every allocator also maintains additional state information  $S_k$ , e.g. round-robin pointers.

At the beginning of a cell slot, the VOQ cell counters are incremented according to the new arrivals  $C_{ij}$  communicated by the line cards:  $L_{ij} \leftarrow L_{ij} + C_{ij}$ . For every  $i, j$ , we define a boolean pre-filter function  $B_{ij}^k$ ,

$$B_{ij}^k \triangleq (\exists i', m_{i'j}^k) \vee (\exists j', m_{ij'}^k), 0 \leq i', j' < N, \quad (1)$$

which expresses whether  $Q_{ij}$  has already been matched in  $\mathcal{M}_k$ , and a boolean request variable  $R_{ij}^k$ ,

$$R_{ij}^k = \begin{cases} \text{true} & \text{if } Q_{ij} \text{ submits a request to } A_k, \\ \text{false} & \text{otherwise.} \end{cases} \quad (2)$$

In general, the submission process, and therefore  $R_{ij}^k$ , can be specified as a function of  $\mathcal{M}_k$ ,  $L_{ij}$ ,  $k$ , or other variables. In Sections II-C through II-E, we will explore three variants for  $R_{ij}^k$ .

The requests as given by  $R_{ij}^k$  are submitted to the allocators, and each allocator  $A_k$  first filters the requests. Any request  $R_{ij}^k$  for which  $B_{ij}^k$  holds is ignored because  $i$  or  $j$  are already matched:

$$R_{ij}^{k,*} \leftarrow R_{ij}^k \wedge \neg B_{ij}^k. \quad (3)$$

Then, each allocator  $A_k$  independently computes a new bipartite matching  $\mathcal{N}_k = [n_{ij}^k]$  on the filtered requests using some

algorithm  $\mathcal{A}$ . The state  $S_k$  is updated. Note that the edges in  $\mathcal{N}_k$  do not share any vertex with any edges in  $\mathcal{M}_k$ , because the corresponding requests have been filtered out. Also note that the requests are non-persistent, i.e., when a request is not granted, it can be resubmitted to another allocator, unlike the scheme of [9], in which a request remains associated with a specific allocator until granted.

After the allocation, the number of new grants for every VOQ is computed:<sup>2</sup>

$$G_{ij} \triangleq \sum_{k=0}^{K-1} \mathbf{1}(n_{ij}^k). \quad (4)$$

Depending on  $R_{ij}^k$ , the number of requests to all allocators and therefore also the number of grants for a given VOQ may exceed the number of pending requests,  $G_{ij} > L_{ij}$ . Accordingly, we apply a boolean post-filter function  $F_{ij}^k$  to modify (some of) the grants:

$$n_{ij}^k \leftarrow n_{ij}^k \wedge F_{ij}^k. \quad (5)$$

Generally,  $F_{ij}^k$  will be a function of  $\mathcal{N}_k$ ,  $\mathcal{M}_k$ ,  $G_{ij}$ ,  $L_{ij}$ ,  $k$ , or other variables. To reduce the implementation complexity, we do not modify the state  $S_k$  of the allocators for filtered-out grants. Note that the filtering operation (5) only applies to the newly matched edges.

We compute the number of grants per VOQ after post-filtering, denoted by  $G_{ij}^*$ . Then, the newly matched edges  $\mathcal{N}_k$  are added to the existing matchings  $\mathcal{M}_k$ :

$$m_{ij}^k \leftarrow m_{ij}^k \vee n_{ij}^k. \quad (6)$$

This operation again yields a valid matching because the edges of  $\mathcal{M}_k$  and  $\mathcal{N}_k$  do not share any vertices, as pointed out above. The arbiter issues grants according to matching  $\mathcal{M}_0$  (allocator 0), which is the last in the pipeline.

Before the next time slot, the counter  $L_{ij}$  is updated:  $L_{ij} \leftarrow \max(L_{ij} - G_{ij}^*, 0)$ , and finally the allocator state is shifted: for  $0 \leq k < K-1$ ,  $\mathcal{M}_k \leftarrow \mathcal{M}_{k+1}$  and  $\mathcal{M}_{K-1} \leftarrow [0]$ . Note that for reasons of fairness the state  $S_k$  is not shifted.

The above description defines a broad class of pipelined matching algorithms according to the choice of the  $R_{ij}^k$  and  $F_{ij}^k$  functions. In particular, we distinguish two subclasses according to whether post-filtering is required. A sufficient, but not necessary condition to avoid post-filtering is to limit the number of requests per VOQ to  $L_{ij}$ :

$$\forall i, j : \sum_{k=0}^{K-1} R_{ij}^k \leq L_{ij}. \quad (7)$$

### C. Method 1: Broadcast requests, selective post-filtering

An important design objective of the methods we introduce here is to keep them simple. Specifically, we only consider methods that do not take  $\mathcal{M}_k$  into account in the request function  $R_{ij}^k$ , which simplifies the hardware implementation. We also keep the post-filter function as simple as possible

by making it independent of  $\mathcal{M}_k$  (Sec. II-C), by omitting it altogether (Sec. II-D), or by satisfying (7) (Sec. II-E).

Method 1 broadcasts requests to all allocators for every non-empty VOQ and cancels any excess grants as defined by (8) and (9), respectively:

$$R_{ij}^k \triangleq (L_{ij} > 0), \quad (8)$$

$$F_{ij}^k \triangleq ((G_{ij} \leq L_{ij}) \vee (k = 0)). \quad (9)$$

In practice, this means that if  $G_{ij} > L_{ij}$ , all grants for  $Q_{ij}$  are cancelled in allocators 1 through  $K-1$  and only the grant of allocator 0, if present, is kept. The rationale behind this filter function is to avoid having to select  $L_{ij}$  out of  $G_{ij}$  grants. Instead, we drop all grants except the one in  $A_0$ , if present.

$R_{ij}^k$  is simple to implement (one comparator), but  $F_{ij}^k$  requires the computation of  $G_{ij}$ , which introduces a centralized operation; see also Sec. V.

### D. Method 2: Broadcast requests, no post-filtering

Eliminating the post-filtering reduces the overall complexity, especially when  $F_{ij}^k$  requires a centralized operation as in Method 1. Omitting the post-filter altogether leads to Method 2 as defined by (10) and (11):

$$R_{ij}^k \triangleq (L_{ij} > 0), \quad (10)$$

$$F_{ij}^k \triangleq \text{true}. \quad (11)$$

However, Method 2 can entail some performance penalties. First, spurious grants may be issued, i.e., a queue may receive more grants than it has cells, leading to a waste of grants. Moreover, excess grants occupy slots in the pipeline that could be used for other, non-empty VOQs, thus potentially degrading performance.

### E. Method 3: Selective requests, no post-filtering

To avoid the possibility of obtaining more grants for a given VOQ than there are requests pending and thus avoid post-filtering, we limit the number of requests to  $L_{ij}$  as defined by (12):

$$R_{ij}^k \triangleq (L_{ij} > k), \quad (12)$$

$$F_{ij}^k \triangleq \text{true}. \quad (13)$$

This eliminates the possibility of excess grants, as only  $L_{ij}$  allocators receive a request. However, in general this method will submit fewer requests early in the pipeline than Methods 1 and 2 do, thus potentially leading to fewer edges being added and therefore to poorer performance.

Note that this method may lead to starvation of short queues ( $L_{ij} < K$ ), see Sec. IV. This starvation cannot occur in Methods 1 and 2 because they broadcast all requests to all allocators.

## III. PERFORMANCE

We study the performance of the three methods described in Sec. II by means of simulation. We use the steady-state simulation method to determine the average throughput and delay with random traffic. Throughput is sampled at the switch

<sup>2</sup> $\mathbf{1}(\cdot)$  is the identity function:  $\mathbf{1}(\text{true}) = 1, \mathbf{1}(\text{false}) = 0$ .

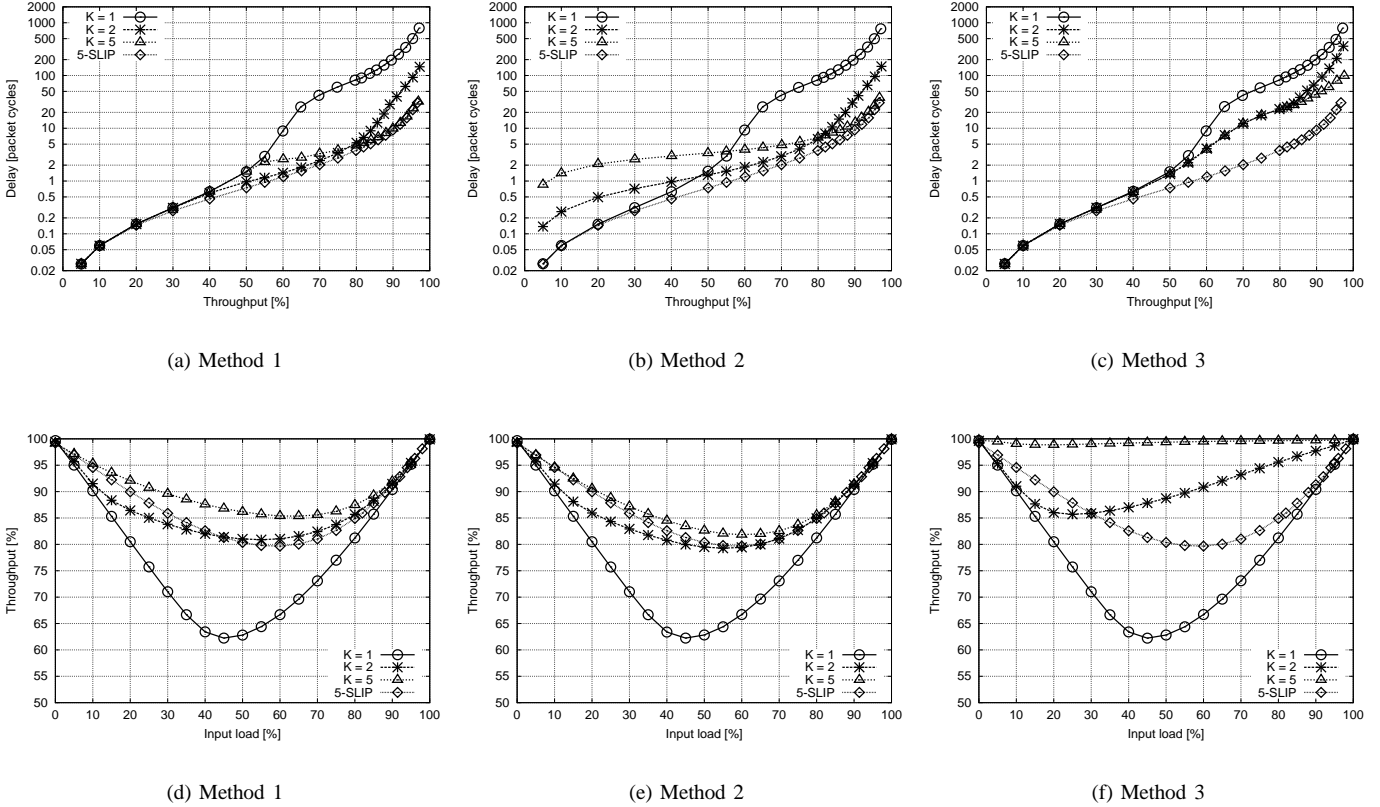


Fig. 4. (a) to (c): Delay vs. throughput with uniform Bernoulli traffic; (d) to (f): Throughput vs. nonuniformity  $w$  with Bernoulli traffic. (a), (d): Method 1 (broadcast requests, selective post-filtering); (b), (e): (Method 2 (broadcast requests, no post-filtering); (c), (f): Method 3 (selective requests, no post-filtering).  $N = 32$ .

gress in every time slot as the ratio of busy to total output ports. Delay is measured end to end and sampled for each packet delivered to the egress. We employ the Akaroa2 [16] parallel simulation management tool to run 15 independent replications of the model and to obtain confidence intervals on the sampled data. The confidence intervals achieved are better than 0.2% with 99% confidence on the throughput, and better than 7% with 95% confidence on the delay. Note that the recorded minimum delay equals zero time slots (cut-through).

#### A. Uniform traffic

Figures 4(a) to 4(c) show the performance for  $N = 32$  with uniform Bernoulli traffic for all three variants of the proposed scheme. For the matching algorithm  $\mathcal{A}$ , every allocator employs one iteration of the dual round-robin matching (DRRM) algorithm [4]. Note that this implies that the number of pipeline stages  $K$  equals the total number of iterations ( $M = K$ ). As a reference, the performance of the  $i$ -SLIP algorithm [3] is also included, with  $\log_2(32) = 5$  iterations *per time slot* (5-SLIP;  $M = 5$ ,  $K = 1$ ). We let  $K$  range from 1 to 5.

Figure 4(a) shows that Method 1 achieves a level of performance that is very close to that of 5-SLIP. Its performance is optimum for  $K = 3$  to 4. In the load range of 50 to 70%,

we observe that the average delay increases very slightly as  $K$  increases. The reason is that cells may be scheduled further into the future than would be necessary and that once a cell has been scheduled, it cannot be transmitted at an earlier time, even if the corresponding ports are idle in an earlier slot.

Method 2 offers about the same performance as Method 1 at loads greater than 60%. However, it clearly has larger latency at low to medium load. As described in Sec. II-D, Method 2 does not filter out excess grants, possibly introducing idle slots that could have been used to transfer waiting cells. However, its behavior at high utilization is almost as good as that of Method 1, because at high loads the queue backlogs  $L_{ij}$  will generally be higher and therefore practically no filtering is needed: if  $\forall i, j : L_{ij} \geq K$ , no filtering is needed at all.

Method 3 exhibits quite poor performance at high loads. Even with  $K = 5$ , the performance still is significantly worse than that of 5-SLIP, especially at more than 50% utilization. This is due to the limited freedom in matching of the allocators that occur early in the pipeline, as these will get fewer requests because of (12).

We have performed the same set of simulations for bursty traffic with an average burst size of 10 packets per burst. These results (omitted due to lack of space) basically exhibit the same trends, but the differences between the three schemes are less

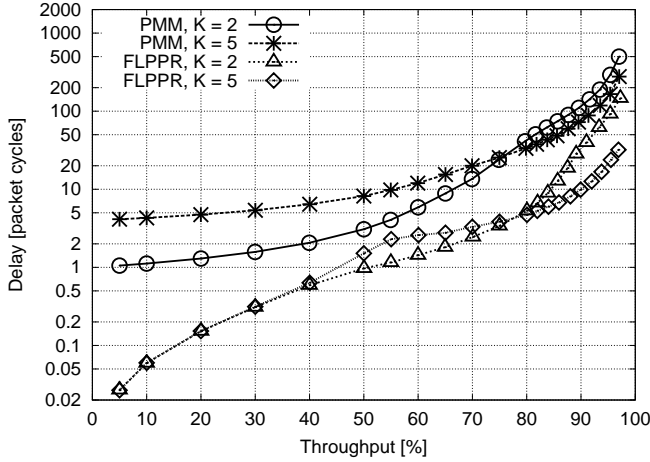


Fig. 5. PMM vs. FLPPR (Method 1), uniform Bernoulli traffic.  $N = 32$ .

pronounced. All schemes perform very close to 5-SLIP.

For comparison, we have simulated the PMM scheme [9] with  $N = 32$ , uniform Bernoulli traffic, and one DRRM iteration per allocator, with  $K$  varying from 1 to 5. In Fig. 5 we compare PMM with FLPPR Method 1. We see that FLPPR achieves a significant reduction in latency at low utilization; PMM incurs a pipelining latency of at least  $K - 1$  time slots. Moreover, FLPPR also achieves significantly lower delay at medium to high utilization. The reason is that the round-robin pointers in FLPPR desynchronize much faster than in PMM; the pointers of a given allocator in PMM are updated only once every  $K$  time slots, whereas those in FLPPR are updated in every time slot. As a result, FLPPR is able to desynchronize the pointers as fast as 5-SLIP.

### B. Nonuniform traffic

To study the performance under nonuniform traffic, we adopt a destination distribution characterized by a nonuniformity parameter  $w$  [17], where  $w = 0$  corresponds to uniform traffic and  $w = 1$  to fully unbalanced, contention-free traffic:

$$\lambda_{ij} = \begin{cases} \lambda \left( w + \frac{1-w}{N} \right) & \text{if } i = j \\ \lambda \frac{1-w}{N} & \text{otherwise.} \end{cases} \quad (14)$$

Here,  $\lambda_{ij}$  represents the traffic intensity from input  $i$  to output  $j$ ,  $0 \leq i, j < N$ ,  $\lambda$  the aggregate offered load, and  $w$  the nonuniformity factor. Note that no input or output is oversubscribed and the traffic is admissible as long as  $\lambda \leq 1$ .

We vary the value of  $w$  from 0 to 1 and measure the throughput achieved at an offered load of 100%. Figures 4(d) to 4(f) show the results for  $N = 32$  and Bernoulli traffic for  $K$  ranging from 1 to 5.

We observe that throughput improves as  $K$  increases, although an increase in parallelism beyond 3 does not bring any significant further improvement. Note that our scheme outperforms 5-SLIP for  $K \geq 3$ . Especially with Method 3 we observe a significant performance improvement as  $K$  increases. With  $K = 5$ , throughput is close to 100% for all  $w$ , which is up to 20 percentage points more throughput

than with 5-SLIP. From [18] we know that to obtain 100% throughput under nonuniform traffic we need a weighted algorithm that takes into account the length of the VOQs (longest queue first—LQF). Method 3 achieves such a weighting effect because it submits more requests for queues that have more backlog, i.e.,  $\sum_k R_{ij}^k$  increases as  $L_{ij}$  increases. Interestingly, Method 3 performs worse than Methods 1 and 2 do for uniform traffic and vice versa for nonuniform traffic. The slightly improved performance of Methods 1 and 2 with respect to 5-SLIP also suggests a weak weighting effect.

**We have performed the same set of simulations (uniform and nonuniform) with correlated traffic with an average burst size of 10 packets per burst. Those results (not shown here) show the same trends as presented above and exhibit no additional throughput limitations.**

## IV. STARVATION

Absence of starvation depends on three factors: the matching algorithm applied by the allocators,  $R_{ij}^k$ , and  $F_{ij}^k$ . Assuming that the selected matching algorithm is starvation-free, such as SLIP or DRRM, we must ensure that  $R_{ij}^k$  and  $F_{ij}^k$  do not introduce starvation. To see how  $R_{ij}^k$  can induce starvation, consider for example Method 3 with  $K = 2$  and a given VOQ  $Q_{ij}$  for which  $L_{ij} = 1$ . This VOQ is never allowed to submit a request to  $A_1$  because  $R_{ij}^1 = (L_{ij} > 1)$  does not hold. If at every time slot there is at least one other input  $i'$  with more than one cell for output  $j$ , the slot for output  $j$  will always be granted to input  $i'$  by  $A_1$ , so that input  $i$  will never be able to obtain a grant for output  $j$  from  $A_0$  after  $M_0 \leftarrow M_1$ .

A trivial example of  $F_{ij}^k$  introducing starvation is when  $F_{ij}^k = \text{false}$  such that no request will ever be granted.

Given that the matching algorithm employed by all allocators is starvation-free, (15) and (16) are sufficient conditions to guarantee that the pipelined version is also starvation-free:

$$L_{ij} > 0 \Rightarrow R_{ij}^{K-1}, \quad (15)$$

$$G_{ij} > 0 \Rightarrow \exists k : F_{ij}^k. \quad (16)$$

The proof is straightforward. As long as  $L_{ij} > 0$ ,  $Q_{ij}$  will persist in submitting a request to  $A_{K-1}$ . As  $\mathcal{M}_{K-1}$  is always empty at the beginning of every time slot and the matching algorithm of  $A_{K-1}$  is starvation-free, this request will eventually be granted, so that no VOQ will remain unserved indefinitely. In practice, (15) means that as long as there are any outstanding requests for  $Q_{ij}$ , a request will be issued at least to  $A_{K-1}$ .

Equation (16) requires that if there are any grants for  $Q_{ij}$  there is at least one  $F_{ij}^k$  that evaluates to true. This implies that not all new grants are filtered out and therefore  $F_{ij}^k$  cannot introduce starvation. Note that Methods 1 and 2 satisfy (15), whereas Methods 2 and 3 satisfy (16).

Strictly optimizing for minimum latency may introduce starvation. From a latency point of view, we want to submit requests as close to  $A_0$  as possible because these, if granted, will be issued sooner. On the other hand, absence of starvation can be guaranteed if requests are (also) issued to  $A_{K-1}$ .

## V. IMPLEMENTATION

Figure 6 presents a block diagram of a possible implementation of the general method.  $A_{K-1}$  does not need a pre-filter function because initially its matching is always empty. In general, the pre- and post-filter sections, the adders, and the counters are required per input-output pair.

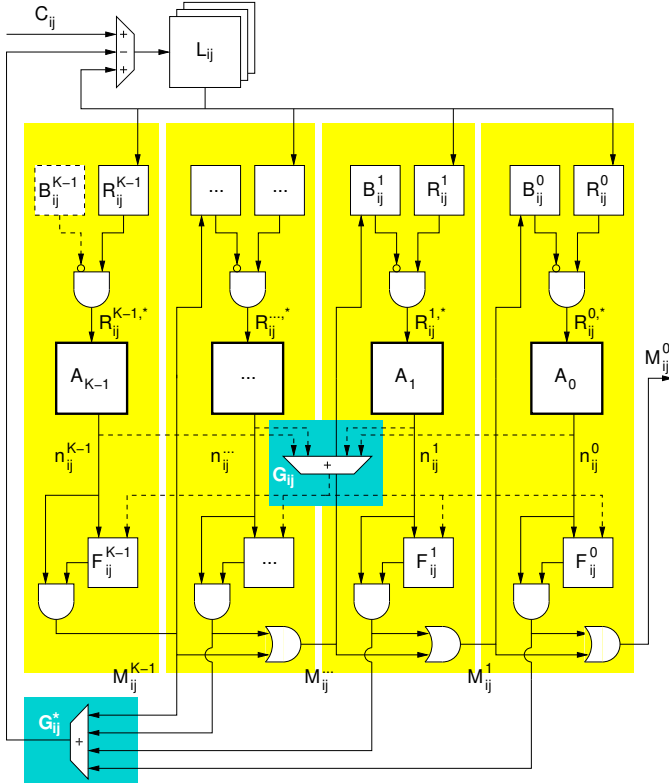


Fig. 6. FLPPR block diagram.

In Fig. 6 we can clearly identify the individual operations from Eqs. (3), (4), (5), and (6). The light-gray boxes illustrate how the implementation could be distributed over multiple ASICs or FPGAs with minimal communication in between them. The dark-gray boxes represent units that are centralized, specifically the adders to compute  $G_{ij}$  and  $G_{ij}^*$ . These are connected to all stages of the pipeline.

## VI. CONCLUSION

We have proposed a novel method named FLPPR to pipeline iterative matching algorithms used in unbuffered crossbar switches. The main advantage is that it makes the minimum latency equal to one time slot, which is independent of the number of pipeline stages, because requests may enter the pipeline at any point. The method also offers high maximum utilization, because the parallel pipelining allows a large number of iterations. These properties make it attractive for a wide range of applications, e.g. for parallel computer interconnects, where low latency is the prime objective, as well as for communications, where high utilization is more important.

The novelty of the proposed method lies in the combination of *parallel* requests to multiple allocators with a *pipeline* in which every allocator expands the matching of the preceding allocator. Furthermore, the addition of special request and post-filter functions allows selective requests and grants to optimize the overall performance characteristics.

The method comprises a broad class of pipelining algorithms based on the aforementioned request and post-filter functions. We have examined three instantiations (Methods) of this class and outlined the performance vs. complexity tradeoffs involved in their design. The performance simulation results with  $N = 32$  have shown that, with uniform traffic, Method 1 with  $K = 5$  achieves delay-throughput characteristics that are very close to those of 5-SLIP with  $K = 1$ , whereas with nonuniform traffic Method 3 clearly outperforms the other methods as well as 5-SLIP with  $K = 1$ , owing to its inherent weighting property.

In future work, we will investigate more sophisticated request and filter functions to combine the uniform-traffic performance of Method 1 with the nonuniform-traffic performance of Method 3, and address the starvation issues of Method 3. Note also that the different allocators need not necessarily execute the *same* algorithm. This possibility is a subject of further investigation.

## REFERENCES

- [1] J. Hopcroft and R. Karp, "An  $n^{5/2}$  algorithm for maximum matching in bipartite graphs," *Soc. Ind. Appl. Math. J. Computation*, vol. 2, no. 4, pp. 225–231, Dec. 1973.
- [2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for local area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, Nov. 1993.
- [3] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.
- [4] 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.
- [5] H. Chao, "Saturn: A terabit packet switch using dual round robin," *IEEE Commun. Mag.*, vol. 38, no. 12, pp. 78–84, Dec. 2000.
- [6] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, vol. 19, no. 1, pp. 20–28, Jan./Feb. 1999.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] —, "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.
- [11] M. Nabeshima, "Input-queued switches using two schedulers in parallel," *IEICE Trans. Commun.*, vol. E85-B, no. 2, pp. 523–531, Feb. 2002.
- [12] S. Mukherjee, F. Silla, P. Bannon, J. Emer, S. Lang, and D. Webb, "A comparative study of arbitration algorithms for the Alpha 21364 router pipelined router," in *Proc. ACM ASPLOS-X*, San Jose, CA, Oct. 2002, pp. 223–234.
- [13] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. San Francisco, CA: Morgan Kaufmann, 2003.
- [14] Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13–27, Jan. 1993.

- [15] J. S. Turner, "Terabit burst switching," *J. High Speed Networks*, vol. 8, no. 1, pp. 3–16, Jan. 1999.
- [16] K. Pawlikowski, H.-D. J. Jeong, and J.-S. R. Lee, "On credibility of simulation studies of telecommunication networks," *IEEE Commun. Mag.*, vol. 40, no. 1, pp. 132–139, Jan. 2002.
- [17] 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.
- [18] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE INFOCOM '96*, vol. 1, San Francisco, CA, Mar. 1996, pp. 296–302.