Belief-Propagation Assisted Scheduling in Input-Queued Switches

Belief-Propagation Assisted Scheduling in Input-Queued Switches

Shadi Atalla, Davide Cuda, Paolo Giaccone, Marco Pretti
Dipartimento di Elettronica and Consiglio Nazionale delle Ricerche, Politecnico di Torino, Italy

Abstract—We consider the problem of scheduling the transmission of packets in an input-queued switch. In order to achieve maximum throughput, scheduling algorithms usually employ the queue length as a parameter for determining the priority to serve a given queue. In this work we propose a novel scheme to optimize the performance of a preexisting scheduler. Our main idea is to *assist* the scheduling decision, considering "messages" rather than queue lengths. Such messages are obtained by running an iterative parallel algorithm, inspired by a rigorous Belief-Propagation approach. We demonstrate that Belief-Propagation-assisted scheduling is able to boost the performance of a given scheduler, reaching almost optimal throughput, even under critical traffic scenarios.

Index Terms—Scheduling, input queued switches, belief propagation, maximum weight matching problem, router architectures.

1 Introduction

Traffic demand in the Internet keeps growing and the switching architectures must be designed to process a steadily increasing amount of data. One classical reference model for high-speed switching architectures is the input-queued (IQ) switch, where packets are stored in queues, before being transmitted across a bufferless, high-bandwidth switching fabric. Such a scheme allows for maximum switching speed. Indeed, packets are transferred across the switching fabric at a speed equal to the line rate, so that the queues do not require any speedup in their access speed. The latter fact allows one to scale the performance much better than any other alternative queueing structure [1]. In an IQ switch, a scheduling algorithm has to choose, at any time, a set of non-conflicting packets to be transferred across the switching fabric. This task requires to solve a combinatorial optimization problem in a very short time, corresponding to the packet transmission time (e.g., a 64 bytes packet, transmitted at 10 Gbps, lasts 51 ns). Only hardware implementations are capable of meeting such tight time constraints. Note that the IQ switch is also one common architecture to implement routing modules in network-on-chips [2].

The design of scheduling algorithms has stimulated a huge scientific literature, motivated by the following dichotomy. On the one hand, the Maximum Weight Matching (MWM) algorithm could ensure optimal performance, using the queue lengths as weights in the matching computation, but this algorithm is too complex to be implemented directly in hardware, because of its centralized and sequential nature. On the other hand, simpler (heuristic) scheduling algorithms, based on parallel architectures, can be implemented satisfying the time constraints, but they turn out to suffer from heavy performance limitations.

S. Atalla and P. Giaccone are with the Dipartimento di Elettronica, Politecnico di Torino, Italy. D. Cuda and M. Pretti are with Consiglio Nazionale delle Ricerche, Italy.

1.1 The Belief Propagation approach

We propose a novel approach to boost the performance of a preexisting scheduler, whose decisions are based on the queue lengths (as is the case in schedulers mimicking MWM decisions). The main underlying idea relies on the theory of probabilistic inference over graphical models, and specially on a related class of distributed *message-passing* algorithms, generally known as Belief Propagation (BP).

BP was first conceived as a dynamic-programming algorithm for evaluating marginal probabilities for so-called Markov random fields, defined on graphs without loops (trees) [3]. The corresponding algorithms for computing maximum-a-posteriori probabilities, which can be used to solve combinatorial optimization problems like MWM [4], are known as *max-product* or *min(max)-sum* (although they are sometimes called BP as well, as we shall do in this paper). It has been demonstrated that, in most cases, BP and related algorithms work surprisingly well even for graphs with loops, where they can be viewed as iterative algorithms. Such a successful behavior seems to be related to the fact that BP fixed-points are equivalent to minima of an approximate free-energy function (Bethe free energy) for a corresponding thermodynamic system. Note the *Bethe approximation* has been employed in statistical physics since long [5], but the connection with BP is a relatively novel result [6].

Given an undirected graph with weighted edges, the MWM is defined as a matching (i.e., a subset of edges without shared nodes) which maximizes the total weight. Each node of the graph is associated to a constraint, that forbids more than one incident edge to

belong to the matching. Generally, BP can be regarded as an exchange of messages between the nodes of the graph; for the MWM problem, a message sent by a node i to a neighbor node j is, roughly speaking, an estimate of the maximum weight which could be obtained, if the ij edge is included in the matching, neglecting the constraint associated to the j node. The messages are iteratively updated *in a parallel and distributed way* across all the nodes, until convergence. Then, in the matching estimation phase, each node takes a *local decision*, based on the incoming messages, and selects one of the incident edges as a candidate for the MWM. The outcome of these distributed decisions is actually a matching, provided that the MWM is unique and that the message values have reached convergence.

In principle, by means of the BP approach, it is possible to compute the MWM in a completely distributed and parallel manner. This approach was first proposed for an IQ switch in [7]. Unfortunately, even though it has been proved that BP converges and provides exact results for the MWM problem [4], [8], the convergence may be in fact very slow, making the whole method impractical.

1.2 Our contributions

In spite of these limitations, we propose in this paper an alternative scheme, which exploits the information, carried by BP messages, to "drive" some ordinary heuristic scheduler. Our *assisted scheduling* approach is a new scheduling procedure, where BP is used to boost the performance of an iterative matching algorithm. The final matching estimation of BP is replaced by a traditional matching algorithm, to avoid the required convergence of the messages. Our approach does not require any change in the original scheduling architecture and scheduler, since we simply use BP messages, instead of the usual queue lengths, as weights for conventional scheduling algorithms. This solution is implementable natively in hardware, and it actually turns out to support and improve the scheduler performance, at low implementation cost. This choice, motivated by the nature of the BP algorithm itself, has also the considerable advantage of avoiding changes to the numerical representation of the weights in the scheduler hardware.

Furthermore, we propose to exploit the time correlation induced by the queue behavior, namely, the fact that the queue lengths can vary at most by ± 1, for each packet arrival or departure. Accordingly, the optimal solution to the MWM problem at a given time is necessarily correlated with the one at subsequent times. Similar correlations appear between BP messages at adjacent times, which allows us to curb the delay times due to message computation.

The paper is organized as follows. In Sect. 2 we describe the scheduling architecture and introduce the concept of assisted scheduling. In Sect. 3 we describe the classical version of the BP algorithm and our novel version, aimed at improving performance and convergence speed. In Sect. 4 we demonstrate by simulations that our BP-assisted scheduling can actually boost throughput performance. In Sect. 5 we draw some conclusions. A high-level hardware design of the proposed system is described in [9] and in a more detailed technical report [10].

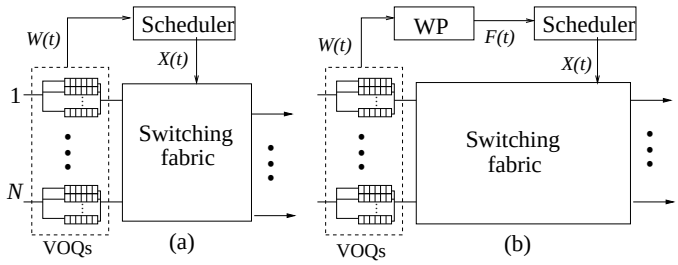

Fig. 1. Input queued switch architecture: (a) standard and (b) equipped with assisted scheduling.

2 System model

We consider a $N \times N$ bufferless, non-blocking switching fabric (e.g., a crossbar), as shown in (a) of Fig. 1. We assume synchronous operations, i.e., time is divided into intervals of fixed duration (timeslots) and the switching fabric transfers data units of fixed size (cells). The timeslot duration is equal to the transmission time of a cell. In the case of variable-size packets, incoming packets are chopped into cells, whereas outputs collect and reorder cells belonging to the same packet, before forwarding to the output interfaces. No output queue is needed, except for reassembly and reordering purposes. We also assume that no speedup is available, i.e., cells are transferred across the switching fabric at the line rate. This allows for high scalability, in terms of number of ports and bandwidth, but imposes the following transfer constraint: when cells from different inputs are destined to the same output, just one cell can actually be transferred. The other cells are stored in the input queues, organized according to the Virtual Output Queue (VOQ) architecture, to avoid the well-known Head of the Line (HoL) blocking problem [11]. In more detail, each input is equipped with N separate FIFO queues (one for each output) of size $Q-1$, each queue storing cells destined to a specific output: a total of N^2 queues is present at the inputs, each one writing and reading cells at the line rate. Let $W(t) = [w_{ij}(t)]$ be a $N \times N$ matrix, whose generic entry $w_{ij}(t)$ is the queue length of the VOQ from input i to output j at timeslot t.

2.1 The scheduler

At each timeslot, a centralized scheduler, shown in Fig. 1, selects a set of HoL cells to be transferred. Such cells must satisfy the aforementioned physical constraints of the switching fabric. A feasible switching configuration can be represented by a matching in a complete bipartite graph of $2N$ nodes, the N left-side nodes representing

the inputs and the N right-side nodes representing the outputs. The edge connecting input i to output j is associated to the corresponding VOQ. Let $X(t) = [x_{ij}(t)]$ be a $N \times N$ matrix, whose generic entry $x_{ij}(t)$ is a boolean variable, equal to 1 iff the scheduler selects a cell to be transferred from input i to output j during timeslot t. These are the decision variables of our problem. They must satisfy the transfer constraints, which can be written as:

$$\sum_{k=1}^{N} x_{ik}(t) \leq 1 \qquad \forall i, t \tag{1}$$

$$\sum_{k=1}^{N} x_{kj}(t) \leq 1 \qquad \forall j, t \tag{2}$$

The evolution of each queue is simply described by $w_{ij}(t+1) = w_{ij}(t) + a_{ij}(t) - x_{ij}(t)$, where $a_{ij}(t)$ is another boolean variable, equal to 1 iff a new cell addressed to j has arrived at i during timeslot t. For the sake of conciseness, in the following we shall omit t from the notation when the context is not ambiguous.

In general, the scheduling decision is based on the state of the queues, obtained by a control path connecting the queueing system to the scheduler. We assume that the control path communicates the actual occupancy state $W(t)$ of all VOQs to the scheduler, as shown in (a) of Fig. 1. If the weight of the edge from input i to output j is equal to w_{ij}, it is well known [12] that computing the MWM on the bipartite graph, weighted by the queue lengths, guarantees 100% throughput under any admissible Bernoulli i.i.d. traffic (this result has also been extended to more general arrival processes [13]). Unfortunately, MWM requires $O(N^3)$ operations and it cannot be efficiently implemented in hardware, due to its sequential nature, difficult to parallelize. Indeed, to the best of our knowledge, MWM has never been implemented on commercial chipsets but it has inspired a large literature investigating heuristic algorithms, simple to implement in hardware, even if not throughput-optimal. Notably, iSLIP [1] is an iterative algorithm in which N arbiters concurrently select a subset of edges to compute the matching, in a parallel and distributed way. Due to its parallel nature, also compatible with pipeline processing (as discussed in [14]), it has been implemented on a single chip and used in a commercial core router. iSLIP approximates a maximal *size* matching, as it does not take into account the queue lengths. A better approximation to the MWM is obtained by iLQF [15], which can be designed with the same architecture as iSLIP, but with a larger control information exchanged by the arbiters; iLQF will be described in Sect. 4.2. Many other algorithms have been proposed to approximate the MWM or to achieve similar performance, by giving priority to longer queues. As mentioned in the introduction, the crucial issue is that optimal or nearly optimal algorithms are difficult to be implemented in hardware; conversely, implementable algorithms cannot achieve 100% throughput under generic traffic scenarios.

2.2 Assisted scheduling

Motivated by this dichotomy, we propose a method to boost the performance of a generic scheduler, which takes decisions on the basis of a queue length matrix $W(t)$. As previously mentioned, this method can be compatible with pre-existing scheduler implementations, since it just adds a functional module between the VOQ and the scheduler itself, as shown in (b) of Fig. 1. This module, which we shall denote as *Weight-Processing* (WP), is based on the update rules of BP-MWM, which is a well known BP algorithm (described in details in Sect. 3.1), designed to compute the MWM for the given set of weights (queue lengths). Since the decision variables are boolean (two possible values), the messages are *scalar* and estimate the *difference* between the maximum weights that can be obtained by the two possible choices of each decision variable. The scheduler runs in the usual way, but, instead of being fed by the queue length matrix $W(t)$, it is fed by a message matrix $F(t)$, whose entries, computed by the WP module, have the same numerical format as those of $W(t)$. Roughly speaking, it is like if WP were "cheating" the scheduler, providing "fake" queue lengths. The reason why such quantities actually boost the scheduler performance should be clearer from the next section, where we describe the message processing in full detail. From a practical point of view, the interesting fact is that the WP module can be implemented independently of the scheduler and that it is amenable to an efficient parallel implementation. The latter issue goes beyond the scope of the current brief contribution, but it is thoroughly discussed in the aforementioned technical report [10].

3 The Scheduling Algorithm

The classical algorithm to compute the MWM is based on a flow augmentation process, and the resulting computational complexity is $O(N^3)$, with optimized data structures. As previously mentioned, it is a sequential algorithm, difficult to parallelize and implement in hardware. On the other hand, BP-MWM provides an alternative rigorous approach to compute the MWM, natively based on parallel computing. Sect. 3.1 describes the basic BP-MWM algorithm; Sect. 3.3 explains how to integrate it with a preexistent scheduler, and provides the main algorithmic contribution of our work.

3.1 Basic Belief-Propagation for MWM

The BP-MWM algorithm is basically equivalent to the min-sum formulation studied in [4], and is also very similar to the one developed in [8] for the more general b-matching problem. In [9], [10] we also report a theoretical derivation, aimed at explaining the conceptual origin of the algorithm, without resorting to previous knowledge from the theory of graphical models.

The pseudocode of BP-MWM is reported in Fig. 2. The algorithm is based on three phases: Initialization,

BP-MWM: *Input*: $W = [w_{ij}]$; *Output*: $X = [x_{ij}]$
// Initialization
for all i, j
$x_{ij} = 0$, $f^0_{i\to j} = w_{ij}$, $b^0_{j\to i} = w_{ij}$
// Update phase
while (messages not yet converged)
for all i, j

$$f^n_{i\to j} = \max\{0, w_{ij} - \max_{k\neq j} b^{n-1}_{k\to i}\} \quad (3)$$

$$b^n_{j\to i} = \max\{0, w_{ij} - \max_{k\neq i} f^{n-1}_{k\to j}\} \quad (4)$$

// Estimate phase
for all j // at each output
$\hat{\imath} = \arg\max_i f^{I_{\text{conv}}}_{i\to j}$; if $f^{I_{\text{conv}}}_{\hat{\imath}\to j} > 0$ then $x_{\hat{\imath}j} = 1$

Fig. 2. Pseudocode of the basic BP-MWM algorithm.

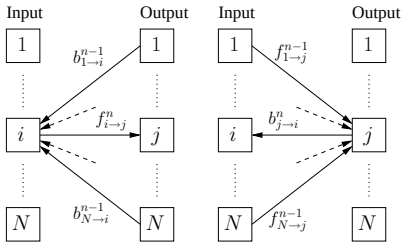

Fig. 3. Parallel message update in WP.

in which all the messages and decision variables are initialized; Update, in which the messages are updated iteratively; Estimate, in which the decision variables are fixed, depending on the message values obtained at the end of the Update phase. The algorithm takes as input a weight matrix W, and returns the boolean matrix X, which represents the matching. The quantities on which it operates are a matrix $F = [f_{i\to j}]$ of "forward messages", from any input i to any output j, and an analogous matrix $B = [b_{j\to i}]$ of "backward messages" (from outputs to inputs). Both matrices are initialized as $F = B = W$, whereas all the entries of X are initialized at 0.

During the Update phase, all N^2 messages are iteratively updated, following the non-linear rules (3) and (4), which characterize the BP-MWM algorithm. The specific message dependencies allow one to parallelize the update process at each iteration n, according to the scheme shown in Fig. 3. A given forward message from input i to output j (namely, $f^n_{i\to j}$) is generated by backward messages, computed at the previous iteration $n-1$, coming from all the outputs but j (namely, $b^{n-1}_{k\to i}$, with $k \neq j$). The same holds for backward messages, generated by forward messages. The iterative Update phase ends in I_{conv} iterations (a-priori unknown), when the messages converge to fixed values. Let us note that the convergence criterion adopted in [4] is slightly looser, but this fact will be irrelevant to our approach, which does not require to reach convergence, as discussed below.

During the Estimate phase, each output j determines the input $\hat{\imath}$ corresponding to the maximum forward message $f_{\hat{\imath}\to j}$. If such a message is nonzero, then the output sets $x_{\hat{\imath}j} = 1$ (i.e., $\hat{\imath}$ is connected to j in the matching). Note that it is also possible to define an alternative (but perfectly equivalent) estimate phase at each input i, choosing the maximum backward message.

3.2 Convergence of BP-MWM

In principle, the Update phase of BP-MWM is not guaranteed to converge on a generic loopy graph (as in our complete bipartite graph), so that also the x_{ij} assignments, computed locally by each output during the Estimate phase, may not represent a feasible matching. In fact, the recent work by Bayati, Shah, and Sharma [4] has proved both convergence and exactness of BP-MWM for precisely our problem (namely, MWM on a complete bipartite graph), under the assumption that the MWM is unique. Even though this result is extremely interesting from a theoretical point of view, the actual implementation of a pure BP algorithm still poses several problems. First of all, the number of iterations I_{conv} is still $O(N^3)$ as in the classical flow-augmentation based algorithm for MWM, but I_{conv} is not a-priori known, and turns out to depend on the weights [4]. Note that, before reaching convergence, the messages cannot be directly used for the Estimate phase, since the distributed choice may lead to unfeasible solutions, not satisfying the matching constraints (1)-(2). Moreover, [4] proves that the convergence time scales also as $O(\epsilon^{-1})$, where ϵ is the difference between the weight of the maximum weight matching and the one of the second-ranked (in terms of total weight) matching. The lower this difference, the longer BP-MWM may take to converge; when the MWM is not unique, BP-MWM may not converge at all. We claim that such a behavior negatively affects our scenario, since, in order to maximize throughput, the MWM scheduler tends to "equalize" queue lengths, likely giving rise to a large number of different matchings with similar or equal weights. Alternatively, one could satisfy the MWM uniqueness requirement, by adding some random "noise" to the entries of W before feeding it to the scheduler, but such a possibility would pose some extra challenges, concerning the noise sample generation and the numerical representation of the messages.

3.3 Belief-Propagation assisted scheduling

Hereafter, we discuss the techniques we propose to implement efficiently the WP module and assist the scheduling decision. Basically, WP runs only the Update phase of BP-MWM, whereas the Estimate phase is "left" to the ordinary scheduler. At odds with BP-MWM, which computes a matching, WP just computes N^2 messages, that will be used by the scheduler (instead of the N^2 VOQ lengths) to compute the final matching. Recall that, when messages converge, the largest message toward each output corresponds to the exact MWM. Interestingly, most of the greedy algorithms for

WP: *Input*: $W(t) = [w_{ij}(t)]$, I; *Output*: $[f^I_{i\to j}(t)]$
// Memory
for all i, j
  $f^0_{i\to j}(t) = f^I_{i\to j}(t-1)$, $b^0_{j\to i}(t) = b^I_{j\to i}(t-1)$
// Update phase
for $n = 1 \dots I$ // run for I iterations
  for all i, j
    if $w_{ij}(t+1) \neq w_{ij}(t)$ // self-asynchronous update
      $f^n_{i\to j}(t) = \max\{0, w_{ij}(t) - \max_{k\neq j} b^{n-1}_{k\to i}(t)\}$
      $b^n_{j\to i}(t) = \max\{0, w_{ij}(t) - \max_{k\neq i} f^{n-1}_{k\to j}(t)\}$
    else
      $f^n_{i\to j}(t) = f^{n-1}_{i\to j}(t)$, $b^n_{j\to i}(t) = b^{n-1}_{j\to i}(t)$

Fig. 4. Pseudocode of WP.

approximating the MWM would choose such edges, if the weights were given by F instead of W. In particular, the Estimate phase of BP-MWM is identical to the Grant phase of the iLQF scheduler [15], where the output arbiters grant the requests corresponding to the largest weight. Hence, we can claim:

Property 1: If the MWM is unique, BP-assisted iLQF, running with the edge weights given by $F = [f^{I_{\text{conv}}}_{i\to j}]$, computes exactly the MWM.

In order to bound the computation latency due to the WP module, we fix the number of iterations I, usually at a very small constant value ($I = 2$ or 3), so that the Update phase of BP-MWM does not converge ($I \ll I_{\text{conv}}$). As a consequence, the ordinary Estimate phase of BP-MWM may return an unfeasible matching, whereas a heuristic scheduler like iLQF is still able to compute a feasible one, even if it is an *approximation* of the MWM. As I grows, the performance of BP-assisted iLQF will approach the MWM, and, for $I \geq I_{\text{conv}}$, the scheduler will behave exactly as the MWM (thanks to Prop. 1), but at the cost of an unbearable computation latency.

Let us remark that the computational complexity of WP is $O(N)$ for each iteration and for each input or output node. The overall complexity remains $O(N)$, thanks to the fixed number of iterations I and to the possibility of a parallel implementation. This is much smaller than $O(N^3)$, experienced by MWM because of its intrinsic serial nature. The low complexity of WP is also compatible with an efficient hardware implementation, as discussed in our technical report [10].

In the following, we propose three complementary techniques (message memory, self-asynchronous update, and integer message representation), aimed at improving the message convergence (i.e., to decrease I_{conv}) and simplifying the WP hardware implementation. The pseudocode reported in Fig. 4 shows the final scheme proposed. Sect. 4 will show that the combination of all these techniques is crucial to achieve maximum throughput.

3.3.1 Message memory

As previously mentioned, the occupancy of each queue can vary at most by ± 1, due to arrivals and departures: $|w_{ij}(t+1) - w_{ij}(t)| \leq 1$, $\forall t$. This means that the queue state exhibits a strong time correlation, which is necessarily reflected in the message dynamics. For this reason, WP uses the last messages computed at the previous timeslot as initial values for the Update phase:

$$f^0_{i\to j}(t) = f^I_{i\to j}(t-1) \qquad b^0_{j\to i}(t) = b^I_{j\to i}(t-1)$$

Note that this technique provides a much "richer" memory of the past queue states, with respect to the pointers adopted in iterative matching algorithms [1], [15].

3.3.2 Self-asynchronous update

The BP-MWM algorithm, reported in Fig. 2, assumes synchronous and parallel message updates, which is also implementable in hardware. Nevertheless, early studies on BP algorithms [16] have shown that asynchronous updates (i.e., messages updated in a random sequential order) are beneficial for convergence. Unfortunately, an asynchronous implementation is very difficult to be obtained in hardware, since it requires $O(N^2)$ sequential iterations. WP *mimics the asynchronous behavior* by updating, in the usual synchronous way, just a subset of messages, chosen on the basis of the arrival process, namely, messages corresponding to queues whose length changed at the previous timeslot. Hence, at most $2N$ messages are updated for each timeslot. In some sense, this technique exploits the arrivals themselves as a randomness source. This also emphasizes the memory effect, enhancing the time correlation between messages at different timeslots.

3.3.3 Integer message representation

We have already mentioned that, in BP-MWM, some "noise" must be added to the weights, in order to guarantee the MWM uniqueness and the algorithm convergence [4]. In general, this is not easy to obtain, because one has to generate random samples, which also need to be represented as real numbers. Fortunately, in our architecture, BP is not required to converge, whence WP does not need to add noise, because, in case of multiple optimal solutions, arrivals in future timeslots are expected to change the MWM and provide an effect similar to random noise. The main technical consequence is that, without noise, the messages are integer numbers. Furthermore, according to the update rules of Fig. 2, the following bounds hold:

$$0 \leq f_{i\to j} \leq w_{ij} \qquad 0 \leq b_{j\to i} \leq w_{ij}$$

implying that the numerical range of the messages is precisely the same as that of the queue lengths:

Property 2: In WP, $F(t), B(t) \in \{0, 1, 2, \dots, Q\}$.

In the assisted-scheduling approach, this detail is of great importance, because it allows us to represent the messages with the same number of bits as the queue lengths.

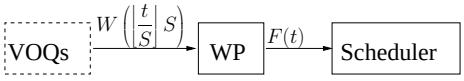

Fig. 5. Slow-BP assisted algorithm: The scheduler processes $F(t)$, which is computed once every S time slots.

This ensures backward compatibility in the control path between the WP module and the scheduler module.

3.4 Slow assisted scheduling

Even though we have fixed the number of iterations I to be small (2 or 3), the running rate of WP can be decoupled from the running rate of the scheduler, proportional to the line rate. Actually, high end routers and switches usually have to take a scheduling decision in at most one time slot, which, depending on the line rate, amounts to few tens of nanoseconds for current 10 Gb/s and 40 Gb/s technologies. As we verify in [9], [10], a FPGA implementation of WP turns out to be compatible with current line rates, but we propose also a "slower" version of the algorithm, providing better scalability, since it can be implemented on slower hardware, or it can cope with future higher line rates, or with large port-count switches.

Fig. 5 displays the high-level architecture implementing the slow version of BP-assisted scheduling. The architecture of the WP module remains unchanged, but the VOQs are sampled every S time slots, i.e. at time $\lfloor t/S \rfloor S$, where S is a suitable *slowing factor*. Accordingly, the BP algorithm is run once every sampling period, taking now S time slots to perform 1 iteration. At each timeslot, the scheduler computes a scheduling decision and avoids serving empty queues. We call this solution *slow-BP* algorithm.

4 Performance Evaluation

In this section, we discuss the performance of BP-assisted scheduling, which we have investigated by means of a discrete-time simulator written in C.

4.1 Simulation settings

We consider both uniform and unbalanced traffic scenarios, that are common reference testbeds for analyzing scheduler performances in IQ switches. Let $\rho \in [0,1]$ be the normalized load at each input port, and λ_{ij} the load from input i to output j ($\Lambda = [\lambda_{ij}]$ is called the traffic matrix). Under uniform traffic, $\lambda_{ij} = \rho/N$. Standard practice among researchers is to require a scheduling algorithm to obtain 100% throughput under such a scenario, which is not considered critical. In order to better evaluate the performance under more critical traffic, we also consider two different unbalanced scenarios, namely, *bi-diagonal* and *log-diagonal*. Under bi-diagonal traffic, $\lambda_{ii} = 2\rho/3$ and $\lambda_{i|i+1|_N} = \rho/3$, where $|x|_N = ((x-1) \bmod N) + 1$, i.e., input port i sends traffic just to output i and $|i+1|_N$. This scenario is a worst case, since just two matchings are "optimal", whereas any other matching leads to significant throughput degradation. No greedy algorithm can achieve maximum throughput under such a critical scenario, as also shown later. Under log-diagonal traffic, one has $\lambda_{ij} \propto \rho 2^{|j-i|}$, i.e., the traffic doubles across each diagonal of the traffic matrix. This scenario can be thought of as a hybrid case between uniform and bi-diagonal.

As far as the maximum queue size is concerned, we set $Q = 1024$ cells. The results are obtained with accuracy better than 1% with a confidence interval of 95%.

4.2 The scheduling algorithms

We consider three different algorithms: iSLIP [1], Longest Queue First (iLQF) [15] and a basic greedy approximation of the MWM, denoted as GWM.

Let us first describe iLQF, which is an iterative algorithm. At the beginning of a timeslot, all inputs and outputs are unmatched. During each iteration, iLQF runs three phases, denoted as *Request*, *Grant*, and *Accept*. During the Request phase (from inputs to outputs), each unmatched input sends all its queue lengths as requests to the corresponding outputs. During the Grant phase (from outputs to inputs), if an unmatched output receives requests, it sends a grant to the input corresponding to the longest queue (contentions are solved randomly). In the Accept phase (from inputs to outputs), if an unmatched input receives grants, it selects the output associated to the longest queue (contentions are solved randomly). During a timeslot, iLQF runs many iterations; at each iteration the three phases consider only unmatched inputs and outputs. The final matching is guaranteed to be maximal if the number of iterations is N, but it has been shown that, under uniform traffic, $\log_2 N$ are actually sufficient [15]. In the following, we always consider $\log_2 N$ iterations for iLQF.

iSLIP runs almost identically as iLQF, but without taking into account the queue lengths. In other words, iLQF becomes equivalent to iSLIP by setting $w_{ij} = 0$ or $w_{ij} = 1$, if the corresponding queue is, respectively, empty or nonempty. Hence, iSLIP cannot be assisted by WP.

GWM is a centralized scheduling algorithm, which computes maximal matchings in a greedy way. At each iteration, GWM selects the unmatched input-output pair associated to the longest queue; it matches that input-output pair and continues with a new iteration. After N iterations, the final matching is maximal.

As far as notation is concerned, in the following we shall use iLQF and GWM to denote the original algorithms, described above, and WP-iLQF and WP-GWM to denote the corresponding BP-assisted versions.

4.3 Simulation results

In Tab. 1 we report the maximum throughput, normalized to one, obtained under uniform, log-diagonal,

TABLE 1
Maximum throughput achieved with $N = 32$, $\rho = 0.99$.

Traffic	iSLIP	iLQF	WP-iLQF	GWM	WP-GWM
Uniform	0.99	0.99	0.99	0.99	0.99
Log-diagonal	0.83	0.97	0.97	0.97	0.97
Bi-diagonal	0.83	0.87	0.98	0.87	0.98

and bi-diagonal traffic scenarios. As well known in the literature, iSLIP behaves poorly, except in the uniform traffic case. The performances of WP-iLQF and WP-GWM are evaluated with $I = 3$ iterations, in the presence of message memory and self-asynchronous update, as described in Sect. 3.3. The effect of these techniques to the final performance of the system is discussed below. Under uniform traffic, all the schedulers obtain maximum throughput. Under log-diagonal traffic, the performance gain due to BP is negligible, as all the algorithms (but iSLIP) achieve throughput larger than 0.97. Conversely, under bi-diagonal traffic, BP-assisted scheduling is able to boost the performance of both iLQF and GWM by more than 10%. In absolute terms this may be a relevant value for a terabit switch. Note that the performance does not depend significantly on the scheduler. For this reason, in the following we consider only one scheduler, namely WP-iLQF, under a bi-diagonal traffic scenario, which appears to be the most critical and meaningful one.

The top panel of Fig. 6 shows the effect of message memory. It turns out that, almost independently of the number of iterations, memory strongly affects the system performance. This result confirms that the strong time correlation between the messages can be fruitfully exploited. In our simulation, $I = 3$ turns out to be sufficient to obtain nearly maximum throughput. The mid panel of Fig. 6 shows the effect of the different update strategies. As expected, the asynchronous strategy turns out to be the most efficient one, but, as previously mentioned, it is difficult to be implemented in hardware. On the other hand, the synchronous strategy, even though amenable to parallel implementation, behaves poorly, almost independently of the number of iterations. Interestingly, the self-asynchronous strategy turns out to be nearly as efficient as the purely asynchronous one. This fact suggests that the arrival randomness effectively mimics an asynchronous update. Finally, in the bottom panel of Fig. 6, we characterize the performance of WP-iLQF (with message memory and self-asynchronous update) for different switch sizes N. In all the investigated cases, few iterations are needed to get nearly optimal performance: $I = 1$ is enough to improve from 0.87 to 0.95, whereas slightly larger numbers of iterations ($I \geq 3$) reach almost 100% throughput. In the following, we always fix $I = 3$.

WP does not affect negatively the delays. Indeed, in all the scenarios considered so far, the delays for WP-iLQF/GWM have been generally observed to be lower than, or (in the worst case) 1.37 times, the delays of

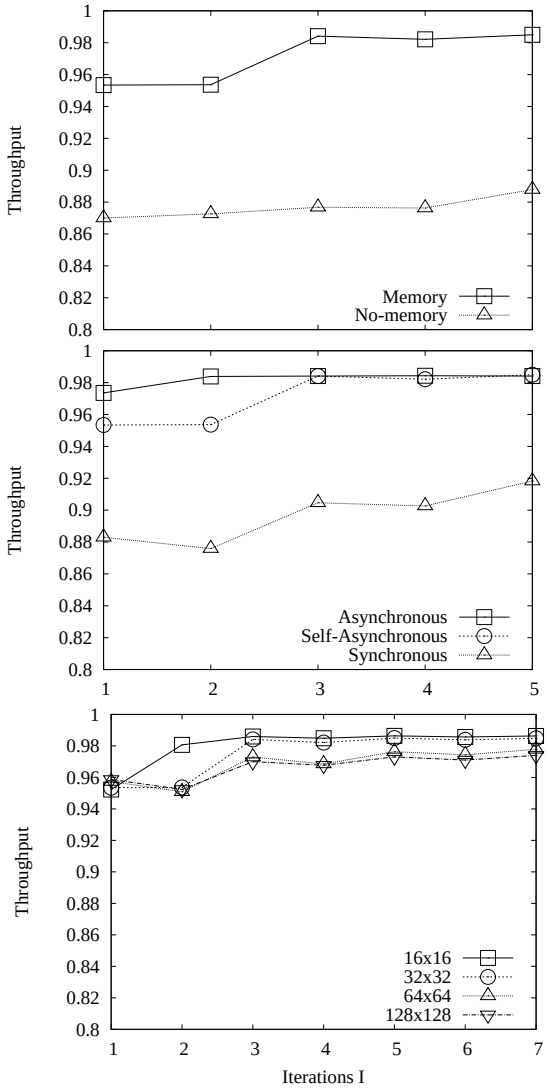

Fig. 6. Throughput achieved by WP-iLQF under bi-diagonal traffic with $\rho = 0.99$ and: (top) $N = 32$, self-asynchronous update, with or without message memory; (middle) $N = 32$, message memory, different update strategies; (bottom) variable switch size, message memory, self-asynchronous update.

TABLE 2
Maximum throughput achieved by slow-WP-iLQF under bi-diagonal traffic, with $N = 32$, $\rho = 0.99$, $I = 3$; message memory, self-asynchronous update.

S	1	2	4	8	16	32	64
Throughput	.953	.951	.946	.938	.927	.916	.905

pure iLQF/GWM. As an example, in Fig. 7 we plot the average delay under bi-diagonal traffic. Note that the flat behavior of iLQF for load larger than 0.9 is due to the finiteness of the queues; according to Tab. 1, loads larger than 0.87 lead to unstable behavior and the delays of iLQF become (theoretically) unbounded.

We have finally analyzed possible throughput degra-

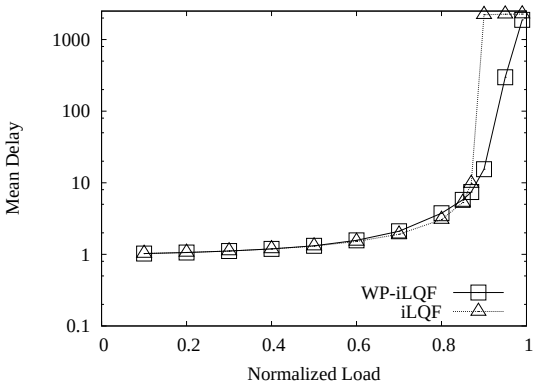

Fig. 7. Average delays for iLQF and WP-iLQF under bi-diagonal traffic, with $N = 32$, $I = 3$; message memory, self-asynchronous update.

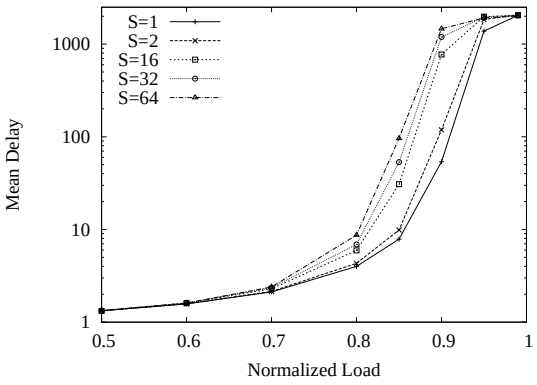

Fig. 8. Average delays for slow-WP-iLQF under bi-diagonal traffic, with $N = 32$, $I = 3$; message memory, self-asynchronous update.

dation, which may occur when iLQF is assisted by slow-BP. Tab. 2 reports some results for the maximum throughput obtained in the usual simulation settings, upon varying the slowing factor S. It turns out that losses become significant only for rather large values of S, such as $S = 32$ or $S = 64$. Fig. 8 shows the corresponding average delays. As S increases, the delay increases, due to the fact that the messages are kept fixed for S time slots. For small values of S, the delay is similar to the one obtained by running WP at each time slot, independently of the load, whereas, for larger S, the difference becomes meaningful, but only at high load values. Of course, the latter effect is due to the "old" state of the queues, processed by the algorithm.

5 Conclusions

We have proposed the Belief-Propagation assisted scheduling as a viable approach to boost the performance of a given scheduler, provided the scheduling decisions are based on the queue lengths. Our scheme is inspired by rigorous Belief-Propagation, enhanced by message memory and self-asynchronous message update. We have shown that the resulting performance is very close to the optimal one. In [9] (with additional details in [10]), we have also investigated the hardware implementation of our approach. The results demonstrate the actual scalability of the approach, leveraging a parallel architecture to compute the messages.

Let us finally remark that the matching problem, considered here, models a generic resource allocation problem occurring in many network scenarios (wireless network control [4], data centers [17]). Therefore, we expect that the impact of our approach might be much wider than the scope of the current paper, and we plan to investigate about these issues in the near future.

References

[1] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[2] S. Vangal *et al.*, "An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[3] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[4] M. Bayati, D. Shah, and M. Sharma, "Max-product for maximum weight matching: Convergence, correctness, and LP duality," *Information Theory, IEEE Transactions on*, vol. 54, no. 3, pp. 1241–1251, Mar. 2008.

[5] H. A. Bethe, "Statistical theory of superlattices," *Proc. R. Soc. A*, vol. 150, no. 871, pp. 552–575, Jul. 1935.

[6] J. Yedidia, "An idiosyncratic journey beyond mean field theory," in *Advanced Mean Field Methods: Theory and Practice*, M. Opper and D. Saad, Eds. MIT Press, Cambridge MA, 2001, pp. 21–35.

[7] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma, "Iterative scheduling algorithms," in *INFOCOM 2007, IEEE*, 6-12 2007, pp. 445–453.

[8] M. Bayati, C. Borgs, J. Chayes, and R. Zecchina, "On the exactness of the cavity method for weighted b-matchings on arbitrary graphs and its relation to linear programs," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 06, Jun. 2008.

[9] S. Atalla, D. Cuda, P. Giaccone, and M. Pretti, "Belief-propagation assisted scheduling in input-queued switches," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, aug. 2010, pp. 7 –14.

[10] ——, "Belief-propagation assisted scheduling in input-queued switches," Technical report available at http://www.tlc-networks.polito.it/public/faculty/paolo-giaccone/publications, 2012.

[11] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *Communications, IEEE Transactions on*, vol. 35, no. 12, pp. 1347–1356, Dec. 1987.

[12] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1260–1267, Aug. 1999.

[13] J. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *INFOCOM 2000, IEEE*, vol. 2, 2000, pp. 556–564.

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

[15] N. W. McKeown, "Scheduling algorithms for input-queued cell switches," Ph.D. dissertation, Berkeley, CA, USA, 1995.

[16] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters," in *ICCV*, 2003, pp. 900–907.

[17] N. Farrington *et al.*, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *ACM SIGCOMM'10*. New York, NY, USA: ACM, 2010, pp. 339–350.