WDM Optical Interconnects with Recirculating Buffering and Limited Range Wavelength Conversion

Zhenghao Zhang, *Student Member*, *IEEE*, and Yuanyuan Yang, *Senior Member*, *IEEE*

Abstract—All-optical communication, in particular, wavelength-division-multiplexing (WDM) technique, has been proposed as a promising candidate to meet the ever-increasing demands on bandwidth from emerging bandwidth-intensive computing/networking applications. However, with current technology, the cost of optical communication, especially the cost of optical buffering and wavelength conversion, remains a major concern for such applications. In this paper, we study WDM optical interconnects that utilize low cost recirculating buffering and limited range wavelength conversion. We first consider the packet scheduling problem in this type of interconnect, and formalize the problem of maximizing throughput and minimizing packet delay as a matching problem in a bipartite graph. We give an optimal parallel algorithm for this problem that runs in $O(Bk^2)$ time, compared to $O((N+B)^3k^3)$ time if directly applied to existing matching algorithms for general bipartite graphs, where N is the number of input/output fibers of the interconnect, B is the number of fiber delay lines, and k is the number of wavelengths. We also consider efficient switching fabric designs for this type of interconnect. We distinguish between the switching fabric connecting the input fibers to the output fibers and the switching fabric connecting the input fibers to the delay lines and show that by adopting the idea of *concentration*, the cost of the latter can be reduced significantly in terms of the number of crosspoints.

Index Terms—Wavelength-division-multiplexing (WDM), optical interconnects, optical packet switching, recirculating buffers, limited range wavelength conversion, concentrators, parallel algorithms, scheduling, matching, bipartite graphs.

1 Introduction and Background

All-optical communication has been proposed as a promising candidate for providing high-speed networking [8], [4], [6] because of the huge bandwidth of optics: A single fiber has a bandwidth of nearly 50 THz [11]. To fully utilize it, the bandwidth of a fiber can be divided into a number of independent channels with each channel on a different wavelength, which is referred to as *wavelength-division-multiplexing (WDM)*. Several technologies have been proposed for WDM, including broadcast and select, wavelength routing, optical packet switching(OPS), and optical burst switching. In this paper, we focus on WDM packet switching as it has better flexibility and better exploitations of bandwidth [8].

In an OPS network, the key component is the optical interconnect (or optical switch) which forwards the packets to their destinations. As in other packet switched networks, the WDM interconnect needs to combat *output contention*. In a WDM interconnect, output contention arises when more than one packets on the same wavelength are destined to the same output fiber at the same time. When this occurs, one will have to either temporarily store some of the packets in a buffer, or to convert wavelengths of the packets to some idle wavelengths by wavelength converters [8]. However, these methods are expensive, since at present, optical random access memory still does not exist and optical buffers are implemented with Optical Delay Lines (ODL) which are very expensive and bulky. Also, optical wavelength converters, if *full range* (i.e., capable of converting a wavelength to any other wavelengths), are very expensive and difficult to implement.

To reduce the cost, instead of allocating dedicated buffers for each output fiber, we can let all output fibers share a common ODL buffer pool [24], [25]. Due to statistical multiplexing, the size of the buffer can be reduced significantly. In a WDM interconnect with shared buffer, a packet that cannot be directly sent to the output fiber is sent to one of the delay lines. After being delayed for some time, the packet will come out of the delay line and attempt to be transmitted again along with the newly arrived packets. If it fails, the packet will be sent to a delay line again to wait for the next round.

The cost can also be reduced by using *limited range* wavelength converters instead of *full range* wavelength converters. Limited range wavelength converter, as the name implies, can only convert a wavelength to a limited number of wavelengths. However, as shown in [9], [5], [10], the performances of networks with limited range wavelength converters are close to those with full range wavelength converters even when the conversion range is small. Therefore, it is a more realistic and cost-effective way to provide wavelength conversion ability.

As an example, Fig. 1 shows a WDM interconnect with recirculating buffer and limited range wavelength conversion. As in [8], [4], [6], we assume that network is time

• *The authors are with the Department of Electrical and Computer Engineering, State University of New York, Stony Brook, NY 11794. E-mail: {zhhzhang, yang}@ece.sunysb.edu.*

Manuscript received 17 May 2004; revised 16 Nov. 2004; accepted 9 June 2005; published online 24 Mar. 2006.
Recommended for acceptance by C. Raghavendra.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0123-0504.

1045-9219/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

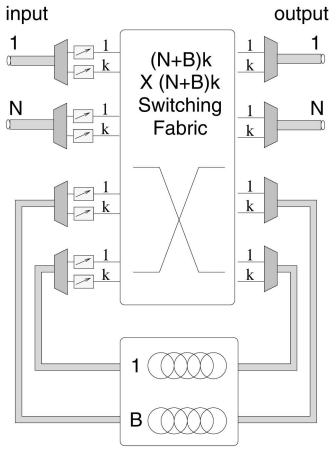


Fig. 1. Optical packet switch with recirculating buffering and wavelength conversion.

slotted and the packets arrive at the interconnect at the beginning of time slots, and the duration of an optical packet is one time slot. Under these assumptions, the interconnect operates in a synchronized manner. The advantage of such a synchronized scheme is that it has better resource utilization than nonsynchronized schemes. The traffic is unicast, i.e., each packet is destined to only one output fiber. The interconnect has N input fibers and N output fibers. Inside the interconnect there are B delay lines, each capable of delaying a packet for one time slot. On each fiber, including the input fiber, the output fiber, and the delay line, there are k wavelengths. Limited range wavelength converters are equipped for the input wavelength channels to the switching fabric. The converters will convert the input wavelength to a wavelength determined by a packet scheduling algorithm. The switching fabric is capable of connecting any one of the $(N+B)k$ inputs, Nk from the input fibers and Bk from the ODLs, to any one of the N output fibers and B ODLs.

We will first consider the problem of packet scheduling in such an interconnect. The goal is to maximize throughput and minimize packet delay. To maximize throughput, as many packets should be sent to the output fibers and ODLs as possible such that minimum number of packets are dropped. In the mean time, to minimize packet delay, whenever possible, a packet should be sent to the output fiber rather than to the ODLs. As will be seen, this problem can be formalized as a weighted matching problem in a bipartite graph and an optimal schedule is an optimal matching in the bipartite graph. However, if directly applying existing matching algorithms for general bipartite graphs, it will take $O((N+B)^3k^3)$ time to find the matching, which is too slow for an optical interconnect. We will show that, due to the limited range wavelength conversion, the bipartite graph will have some nice properties and we can obtain a parallel algorithm called Parallel Segment Expanding Algorithm that runs in $O(Bk^2)$ time to find the optimal schedule.

Matching algorithms for general bipartite graphs were well studied [16]. However, since the bipartite graph considered in this paper exhibits some special properties due to limited range wavelength conversion, it is possible to design new algorithms to speed up the scheduling. Scheduling algorithms for electronic interconnects have also been extensively studied, for example, the iSLIP algorithm [15] for input-buffered interconnects. However, these algorithms cannot be applied to our problem since the recirculating buffer is a shared output buffer. WDM interconnects with dedicated output buffers and full range wavelength conversion were studied and their performance was evaluated with analytical models in [4]. Scheduling in unbuffered WDM interconnects with limited range wavelength conversion was studied in [27], [17], and an optimal scheduling algorithm called the First Available Algorithm was given in [27]. Scheduling in WDM interconnects with dedicated output buffers and limited range wavelength conversion was studied in [6], [18], and an optimal scheduling algorithm called the Scan and Swap Algorithm was given in [18]. Note that the scheduling problem in this paper is quite different from and more complex than those in [27], [18]. Since first, in the case of no buffer or dedicated buffer, scheduling can be carried out for each output fiber independently, but when the buffers are shared, scheduling must be carried out with respect to the entire interconnect. Second, though [27], [18] also formalized the scheduling problems as a matching problem in bipartite graphs, the bipartite graphs are quite different from the bipartite graphs considered in this paper because in [27], [18] the adjacency set of a vertex is always an interval, while, in this paper, this property no longer holds. Therefore, the problem in this paper is much more challenging than those in [27], [18].

Finally, note that in a WDM interconnect, using buffer is at the cost of a larger switching fabric. To reduce the cost, we give a new design of the switching fabric using the idea of *concentration*, which is the second contribution of this paper besides the optimal scheduling algorithm. This new design distinguishes between the switching fabric connecting the input fibers to the output fibers and the switching fabric connecting the input fibers to the ODLs, and can significantly reduce the number of crosspoints of the latter.

The rest of this paper is organized as follows: Section 2 describes the properties of limited range wavelength conversion. Section 3 presents the optimal scheduling algorithm. Section 4 analyzes the complexity of the algorithm and gives some performance evaluation results. Section 5 presents the new switching fabric design. Finally, Section 6 concludes the paper.

2 Wavelength Conversion

All-optical wavelength conversion is usually achieved by conveying information from the input light signal to a probe signal [19], [7]. The probe signal is generated by a tunable laser tuned to the desired output wavelength. The tuning range of the laser is continuous, but under limited range wavelength conversion, it is only part of the whole spectrum due to constraints such as tuning speed, loss, etc.

We can see that a wavelength can be converted to an interval of wavelengths because the tuning range of the laser is continuous and covers an interval of wavelengths. Also, note that if the laser for the conversion of λ_1 can be tuned to λ_3, then the laser for the conversion of λ_2 should also be able to be tuned to λ_3 since λ_2 is closer to λ_3 than λ_1

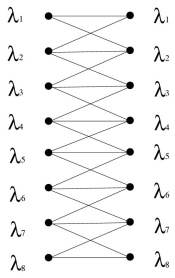


Fig. 2. Wavelength conversion of an eight-wavelength system.

is. These two observations lead to the following two assumptions of wavelength conversion:

Assumption 1. *The wavelengths that can be converted from λ_i for $i \in [1,k]$ can be represented by interval $[Begin(i), End(i)]$, where $Begin(i)$ and $End(i)$ are positive integers in $[1,k]$. Wavelengths belong to this interval are called the* adjacency set *of λ_i.*

Assumption 2. *For two wavelengths λ_i and λ_j, if $i < j$, then $Begin(i) \leq Begin(j)$ and $End(i) \leq End(j)$.*

We call this type of wavelength conversion "ordered interval" because the adjacency set of a wavelength can be represented by an interval of integers, and the intervals for different wavelengths are "ordered." The cardinality of the adjacency set is called the *conversion degree* of the wavelength. Different wavelengths may have different conversion degrees. The *conversion distance* of a wavelength, denoted as d, is defined as the largest difference between a wavelength and a wavelength that can be converted from it.

A bipartite graph can be used to visualize the wavelength conversion. Let the left side vertices represent input wavelengths and the right side vertices represent output wavelengths. λ_i on the left and λ_j on the right are connected if λ_i can be converted to λ_j. Fig. 2 shows such a conversion graph for $k = 8$. The adjacency set of λ_3, for example, can be represented as $[2,4]$. The conversion degree and conversion distance of λ_3 are 3 and 1, respectively.

Note that the assumptions we made about wavelength conversion are very general, only relying on the two facts observed at the beginning of this section. Different wavelengths are allowed to have different conversion degrees and different conversion distances. This type of wavelength conversion is also used in other research works, for example, [20], [21], [17]. Full range wavelength conversion can also be considered as a special case, by letting the conversion degrees for all wavelengths be k.

3 Optimal Scheduling Algorithm—The Parallel Segment Expanding Algorithm

3.1 Formalization of the Scheduling Problem

As mentioned earlier, the goal of the scheduling algorithm should be: 1) To minimize the packet loss, drop as few packets as possible. 2) To minimize the packet delay, send as many packets directly to the output fiber as possible.

This is a typical resource allocation problem and can be formalized as a matching problem in a bipartite graph.

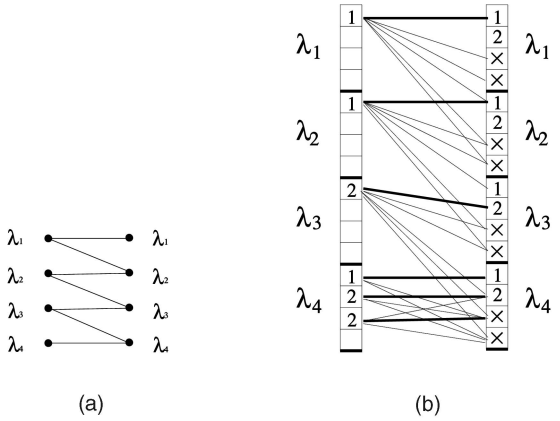


Fig. 3. (a) Wavelength conversion of a four-wavelength interconnect. (b) Packets and wavelength channels in an interconnect with $N = 2$, $B = 2$.

Fig. 3b shows such a graph for a simple interconnect where $N = 2$, $B = 2$, $k = 4$, and the wavelength conversion as defined in Fig. 3a. In this graph, left side vertices represent arrived packets and right side vertices represent wavelength channels. Vertices are arranged according to wavelengths, lower wavelength first. In this example, there are four input (output) channels on each wavelength, two from the input fibers and two from the ODLs. Vertices are represented by boxes. On the left side, label "1" or "2" was put in the box to represent the destination of the packet. A box is left empty if there is no packet on the wavelength channel. On the right side, there are two types of vertices: the *output vertices* and the *buffer vertices*, where output vertices represent wavelength channels on the output fibers and buffer vertices represent wavelength channels on the delay lines. In the figure, an output vertex is labeled as "1" or "2" according to the output fiber it is in, and a buffer vertex is labeled by a cross. A left side vertex, say, a, is connected to a right side vertex, say, b, by an edge (either thin edge or fat edge in this example) if and only if the wavelength channel represented by b can be assigned to the packet represented by a. A necessary condition for a left side vertex to be adjacent to b is that its wavelength must be able to be converted to the wavelength of b. If b is a buffer vertex, all such left side vertices are adjacent to b, regardless of their destinations. However, if b is an output vertex, b is only adjacent to vertices representing packets destined to the output fiber where b is in.

In this bipartite graph, let E denote the set of edges. Any schedule can be represented by a subset of E, E', where edge $ab \in E'$ if wavelength channel b is assigned to packet a. Under unicast traffic, any packet needs only one output channel and an output channel can be assigned to only one packet. It follows that the edges in E' are vertex disjoint since if two edges share a vertex, either one packet is assigned to two wavelength channels or one wavelength channel is assigned to two packets. Thus, E' is a *matching* in G.

To maximize network throughput, we should find a maximum cardinality matching. To minimize the total delay, this matching should cover maximum number of output vertices. To do this, we can assign weight 1 to the output vertices and weight 0 to the buffer vertices, then find

TABLE 1
List of Symbols

b_x	:	the buffer vertex being checked.
G_i	:	subgraph for output i.
M_i	:	current matching in G_i.
M	:	union of M_i for $1 \leq i \leq N$.
$mat[a]$	:	the vertex matched to a
R	:	reachable set of b_x
FS_p^i	:	the p_{th} forward segment in G_i
BS_q^i	:	the q_{th} backward segment in G_i
$breg_i$	:	wavelengths of buffer vertices discovered in G_i
$lreg$	:	wavelengths of newly discovered left side vertices

the optimal matching, which is a matching with maximum cardinality and also maximum total weight. In the example in Fig. 3b, an optimal matching is shown in heavy lines. Next, we will give an outline of our algorithm and introduce some notations. Some of the most frequently used notations are listed in Table 1.

3.2 Outline of the Parallel Segment Expanding Algorithm

For any vertex b, if one of the edges in a matching has one end being b, we say b is *covered* in this matching. Otherwise, we say b is *uncovered* in this matching. Optimal matching in a bipartite graph can be found by the following simple genetic algorithm, which can be called the Matroid Greedy Algorithm [22], [23]. The algorithm starts with an empty set Π. In step s, let b be the vertex with the sth largest weight. The algorithm checks whether there is a matching covering b and all the vertices in Π. If yes, add b to Π, otherwise leave b uncovered. Then, $s \leftarrow s+1$ and repeat until all vertices have been checked. When finished, Π stores weighted vertices that can be covered by an optimal matching.

In an arbitrary bipartite graph with n vertices, to check whether a vertex can be covered along with vertices in Π needs $O(n^2)$ time, thus the time complexity of the matroid greedy algorithm is $O(n^3)$. For our application, it will be as high as $O((N+B)^3k^3)$, where N is the number of input/output fibers, B is the number of delay lines, and k is the number of wavelength channels, which is apparently too slow since the scheduling must be carried out in real time. In the following, we will give a fast optimal scheduling algorithm called the Parallel Segment Expanding Algorithm that solves the problem in $O(Bk^2)$ time. The algorithm is based on the matroid greedy algorithm, however, the running time is greatly reduced due to the following two reasons.

First, in our bipartite graph, vertices have only two types of weights: Output vertices have weight 1 and buffer vertices have weight 0. According to the matroid greedy algorithm, the output vertices should be checked first since they are of larger weights, and then the buffer vertices. Therefore, our algorithm runs in only two phases. In the first phase, it will find a matching that covers the maximum number of output vertices, such that the resulting matching will have the largest weight. In the second phase, it will augment the matching until it covers as many buffer vertices as possible, such that the resulting matching will also have maximum cardinality. Second, our algorithm is run in parallel. We will not use a centralized scheduler that works on the bipartite graph introduced in Section 3.1, rather, we will draw the bipartite graph as the union of N subgraphs, one for each output fiber, and use N processing units to find the matching in parallel.

The subgraphs will be denoted as G_i for $1 \leq i \leq N$. In G_i, the left side vertices represent the packets destined for output fiber i, and the right side vertices include the output vertices on output fiber i and the buffer vertices that are matched to left side vertices in G_i. These buffer vertices are said to be "assigned" to G_i. Unmatched buffer vertices are not shown in any subgraphs. Note that this is an equivalent way for representing the input/output relationship as using the "whole" bipartite graph; nevertheless, this enables us to develop an algorithm that runs in parallel. For example, Fig. 3b can be shown equivalently as in Fig. 4b. It is important to note that output vertices are only adjacent to left side vertices in one particular subgraph while buffer vertices can be adjacent to left side vertices in several subgraphs. We will denote left side vertices as a_i and right side vertices as b_u according to their wavelengths, where i and u are the indices of the vertices. Vertices on lower wavelengths have smaller indices and vertices on the same wavelength are in an arbitrary order.

Note that, in phase one, the parallelism of our algorithm is quite natural since only the output vertices need to be considered while an output vertex in G_i is only adjacent to left side vertices in G_i. As a result, a subgraph has no connections to other subgraphs. For example, if only the output vertices in Fig. 3b are considered, the whole bipartite graph is decomposed into two *isolated* subgraphs shown in Fig. 4a. Therefore, matching the maximum number of output vertices can be achieved by finding a maximum matching for each of the subgraphs *in parallel* and then combining them. However, the problem becomes more complex in phase two since, after assigning buffer vertices to the subgraphs, the subgraphs will not be isolated. For example, a buffer vertex on λ_4 in the example of Fig. 4b can be matched to a_3 in G_2; therefore, it is assigned to G_2, and is

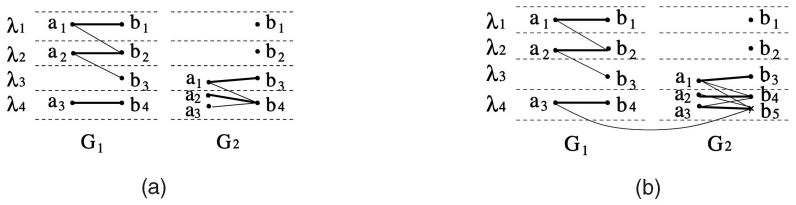


Fig. 4. (a) Fig. 3 is decomposed into two subgraphs when only considering the output vertices. (b) Assigning a buffer vertex on λ_4 to G_2.

TABLE 2
First Available Algorithm

```
for u := 1 to n do
    let a_j be the vertex adjacent to b_u
    not matched yet with the smallest index
    if such a_j exists
        match b_u to a_j
    end if
end for
```

denoted as b_5. Note that it is not only adjacent to vertices in G_2 but also to vertices in G_1. Nevertheless, we can still design an algorithm that runs in all subgraphs in parallel by taking advantage of the properties of the subgraphs.

In phase two, according to the matroid greedy algorithm, we can check the buffer vertices one by one to see whether they can be matched along with all the previously matched vertices. Buffer vertices on lower wavelengths are checked first. We will refer to the buffer vertex being checked as b_x. When checking b_x, the matching in G_i is denoted as M_i, and the union of M_i for $i = 1, 2, \ldots, N$ is denoted as M. If b_x can be matched, we update M to cover b_x; otherwise, M is not changed and we proceed to the next buffer vertex. The details of the method for matching buffer vertices will be described in later sections.

3.3 Phase One—Matching Output Vertices

As explained earlier, in this phase the N subgraphs are *isolated* and we can find a maximum matching for each of them in parallel. Since the method for finding maximum matching is the same for all subgraphs, we will only explain it for one subgraph G_i.

In this phase, the right side vertices of in G_i are all output vertices. Based on the properties of wavelength conversion, G_i has following two properties:

Proposition 1. *The adjacency set of any right side vertex, say, b_u, is an interval and can be represented as $[begin(b_u), end(b_u)]$.*

Proposition 2. *If $u < v$, then $begin(b_u) \leq begin(b_v)$ and $end(b_u) \leq end(b_v)$.*

We call a bipartite graph with Properties 1 and 2 a *request graph*. Maximum matching in request graphs can be found by First Available Algorithm described in Table 2 [27]. This algorithm checks the right side vertices from top to bottom. A right side vertex b_u will be matched to its *first available neighbor*, which is an unmatched left side vertex adjacent to it with the smallest index. The time complexity of this algorithm is $O(n)$, where n is the number of right side vertices, since the loop is executed n times and the work within the loop can be done in constant time. For example, after running the First Available Algorithm, the matchings in Fig. 4a are shown in heavy lines.

It was also shown that a bipartite graph with Properties 1 and 2 has the following property in [27]:

Proposition 3. *If edge $a_i b_u \in E$, $a_j b_v \in E$, and $i < j$, $u > v$, then $a_i b_v \in E$, $a_j b_u \in E$.*

This property can be called the *crossing edge property*, which will be frequently used in proving other properties of request graphs in this paper. It is so called because if $i < j$ and $u > v$, $a_i b_u$ and $a_j b_v$ will appear crossing each other in the request graph. A direct consequence of this property is that there must be a maximum matching of a request graph with no crossing edges since any pair of crossing edges in the matching, say, $a_i b_u$ and $a_j b_v$, can be replaced with $a_i b_v$ and $a_j b_u$ which do not cross each other. Such a matching is called a noncrossing matching, in which the ith matched left side vertex is matched to the ith matched right side vertex.

It can be easily verified that:

Proposition 4. *The matching found by First Available Algorithm is noncrossing.*

Another property of the matching found by the First Available Algorithm, due to the fact that a vertex is always matched to its first available neighbor, is

Proposition 5. *If b_u is matched to a_i, all the left side vertices adjacent to b_u with smaller indices than a_i must be matched to right side vertices with smaller indices than b_u.*

We introduce a more sophisticated example in Fig. 5, which will be used throughout this section to help understand our algorithm. In this example, $N = 4, B = 2, k = 8$, and the wavelength conversion is as defined in Fig. 2. For notational convenience, we use a 1×8 vector called the arrival vector to represent the number of packets destined for an output fiber, in which the ith element is the number of packets destined for this output fiber on λ_i. In this example, the arrival vectors for output fiber 1 to output fiber 4 are $[2, 1, 1, 0, 3, 0, 0, 2]$, $[2, 5, 3, 0, 0, 0, 0, 0]$, $[2, 0, 2, 5, 0, 0, 0, 0]$, and $[0, 0, 0, 0, 0, 0, 0, 0]$, respectively. There is no packet destined for output fiber 4, therefore, only three subgraphs, G_1, G_2, and G_3, were shown. Fig. 5a shows the matching after running First Available Algorithm on each of the subgraphs.

3.4 Phase Two—Matching Buffer Vertices

After phase one, the matching needs to be augmented to cover buffer vertices. The goal is to match as many buffer vertices as possible while keeping the previously matched vertices matched. As mentioned earlier, we will check the buffer vertices one by one. By graph theory, the vertex being checked, b_x, can be matched if and only if there exists an M *augmenting path* with one end being b_x. An M augmenting path is an M-alternating path with both ends being unmatched vertices, where an M alternating path is a path that alternates between edges in M and not in M. More materials about augmenting paths can be found in [28].

Our algorithm also searches for the M augmenting path. However, before searching for the augmenting path, we will first try to use a simpler method to match b_x, which is called "direct insertion." Direct insertion, roughly speaking, is to match b_x in each subgraphs by running the First Available Algorithm in these subgraphs in parallel. b_x can be matched if it can be matched in one of the subgraphs. Note that, if a buffer vertex can be matched by direct insertion, it can also be matched by the augmenting path approach, but the reverse may not be true. We try direct

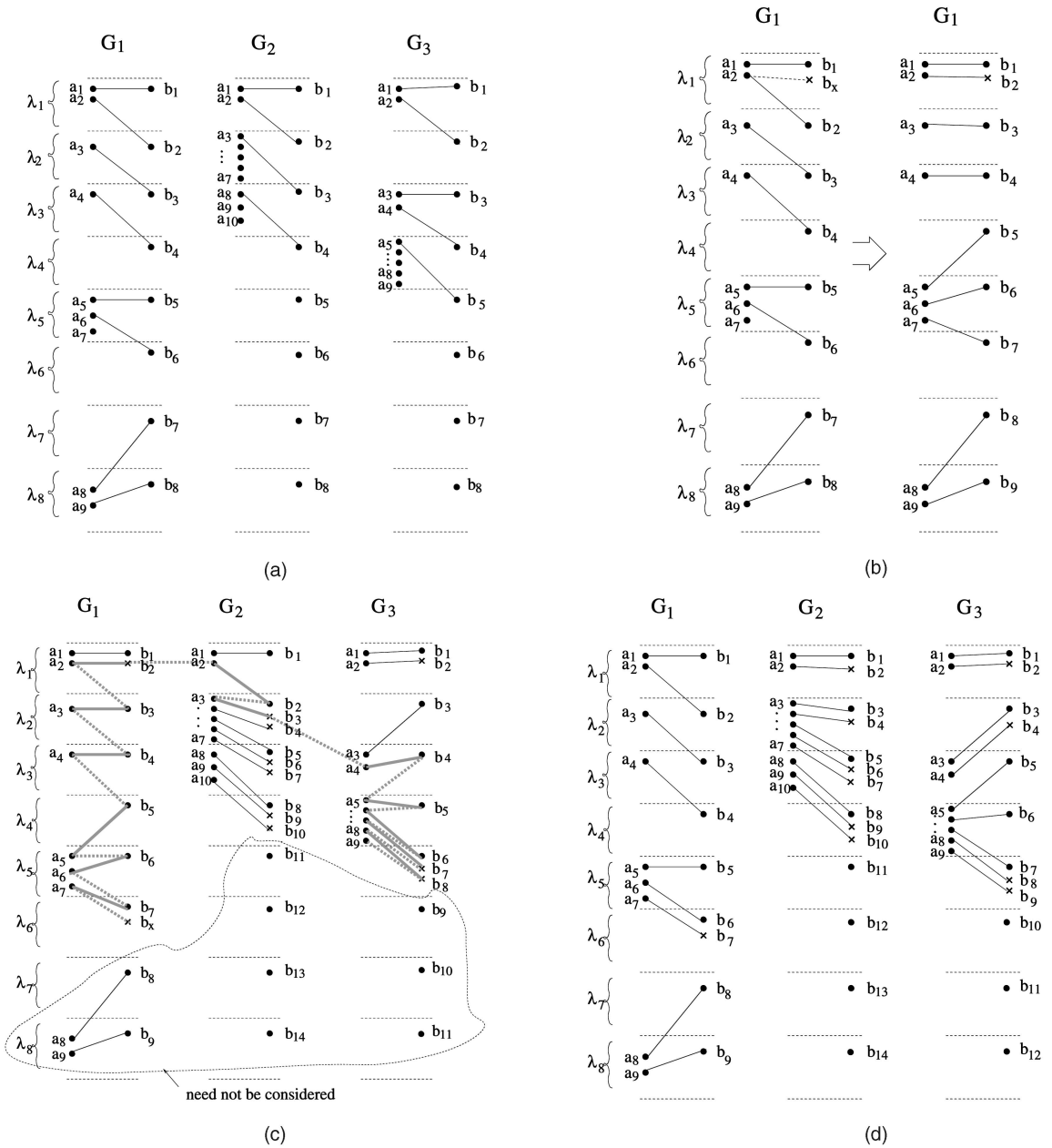


Fig. 5. Matchings of an interconnect with $N = 4$, $B = 2$, $k = 8$, and the wavelength conversion is as defined in Fig. 2. G_4 is not shown here because there is no packet destined for output fiber 4. On the right side, a dot is an output vertex and a cross is a buffer vertex. (a) After matching output vertices by the First Available Algorithm. (b) Direct insertion of a buffer vertex on λ_1 to G_1. (c) An augmenting path for b_x on λ_6. (d) b_x is matched in G_1 and is given an index 7.

insertion first because it is much simpler and, thus, in practice this may greatly reduce the running time of our algorithm.

3.4.1 Matching Buffer Vertices—Direct Insertion

To match buffer vertex b_x by direct insertion in subgraph G_i is to first add a vertex on the same wavelength as b_x to G_i then run the First Available Algorithm. If all the right side vertices in G_i can be matched, including the added vertex, b_x can be matched in G_i and will be assigned to it. Note that this can be done in parallel in all subgraphs. If b_x can be directly inserted to more than one subgraphs, we can arbitrarily pick one.

After a buffer vertex has been assigned to a subgraph, it is given an index. The indices of some right side vertices may also need to be updated. For example, in Fig. 5b, buffer vertex b_x on wavelength λ_1 can be directly inserted into G_1. After assigning it to G_1, it is given an index 2, and the indices of the right side vertices following it are incremented by 1.

3.4.2 Matching Buffer Vertices—An Outline for Augmenting Path Search

As mentioned earlier, even if b_x cannot be directly inserted to any of the N subgraphs, b_x may still be able to be matched. We can try our second method which is to find an M augmenting path. If such an augmenting path is found, we can perform a "flip" operation on the edges in the path, i.e., remove edges that were in M and add in edges that were not in M. The new

matching will cover all previously covered vertices plus the two vertices at the ends of the path.

An example is shown in Fig. 5c. In this figure, a dot represents an output vertex and a cross represents a buffer vertex. Edges in M are represented by solid lines and edges not in M are represented by dashed lines. Let b_x be the buffer vertex on wavelength λ_6. It is not hard to see that b_x cannot be directly inserted in any of the subgraphs. However, it can still be matched because, as shown in the figure, there is an M augmenting path starting at b_x, traversing G_1, G_2, and G_3, and ending at an unmatched left side vertex a_9 in G_3. After the flip operation, the new matching is shown in Fig. 5d. Note that the buffer vertex on λ_1 has been moved from G_1 to G_2 and the buffer vertex on λ_2 has been moved from G_2 to G_3 because according to the new matching they should be assigned to G_2 and G_3, respectively.

It is important to note that, in the augmenting path, some buffer vertices serve as "bridges" to extend the path from one subgraph to another. Also note that to update the matching according to the augmenting path, some buffer vertices are moved from one subgraph to another. The purpose of this moving can be considered as "making room" for b_x in one of the subgraphs. The optimal scheduling algorithm can also be considered as optimally assigning the buffer vertices to the subgraphs such that maximum number of them can be matched.

To find the augmenting path, similar to the algorithms for general bipartite graphs, our algorithm also searches for the *reachable set* of b_x. The reachable set, denoted as R, is defined as the set of vertices that can be reached from b_x via M-alternating paths. If there is an unmatched left side vertex in this set, the augmenting path is found. However, note that the differences between our algorithm and algorithms for more general bipartite graphs are profound. First, our algorithm is executed in all subgraphs in parallel and, as a result, many vertices in different subgraphs can be added to R simultaneously. Second, in the general method, a vertex can be added to R only when it is adjacent to some vertices already in R. In our algorithm, many vertices can be added to R even if they are not adjacent to any vertices currently in R, as long as these vertices are in the same *forward segment* or *backward segment* which will be defined soon. More importantly, this can be done in *constant* time regardless of the number of vertices in the segment, which makes a parallel algorithm possible.

Before presenting the algorithm, due to the importance of the forward and backward segments, in the next section we will first formally define them and describe their properties.

3.4.3 Matching Buffer Vertices—Identifying Forward and Backward Segments

The forward and backward segments are defined within a subgraph; hence, in this section, we consider only one subgraph, say, G_i. First, we assume that M_i is the matching found by running the First Available Algorithm on G_i. Recall that due to the properties of the First Available Algorithm, M_i is noncrossing in G_i.

Note that two types of vertices in G_i do not need to be considered for the purpose of matching a buffer vertex: 1) left side vertices with larger indices than a_h and right side vertices with larger indices than b_u, where b_u is the right side vertex on a wavelength no higher than that of b_x with the largest index and a_h is the vertex matched to b_u; 2) unmatched right side vertices. Type 1 vertices do not need to be considered because when b_x cannot be directly inserted in G_i, left side vertices with larger indices than a_h have to be matched to right side vertices with indices larger than b_u. Type 2 vertices do not need to be considered because they cannot be used to expand the reachable set. Therefore, when searching for an augmenting path for b_x, these vertices will be considered as having been removed from the graph. For example, as shown in Fig. 5c, in G_1, a_8, a_9, b_8, and b_9 have been removed. In G_2, b_{11} to b_{14} have been removed. In G_3, b_9 to b_{11} have been removed.

First, we define *forward segment*. For notational simplicity, for any vertex b, we use $mat[b]$ to denote the vertex matched to it. Note that the index of $mat[b_p]$ is larger than $mat[b_q]$ if $p > q$ since M_i is noncrossing. Now, imagine scanning the right side vertices from $b_1, b_2, \ldots$, until b_v when one of the following conditions is satisfied: 1) b_v is not adjacent to $mat[b_{v+1}]$ or 2) there are some unmatched left side vertices adjacent to b_v. If such b_v is found, b_1 to b_v and all the left side vertices matched to them, plus the unmatched left side vertices adjacent to b_v with smaller indices than $mat[b_{v+1}]$ are called a *forward segment*. After finding the first forward segment, scan from b_{v+1} to find the second forward segment, then the third until reaching b_u, the last right side vertex in G_i. As an example, in Fig. 5c, note that in G_3, though b_1 is adjacent to a_2 ($mat[b_2]$), b_2 is not adjacent to a_3 ($mat[b_3]$). Thus, a_1 to a_2 and b_1 to b_2 should be a forward segment. The second forward segment in G_3 is a_3 to a_9 and b_3 to b_8.

We now describe the properties of forward segments.

Proposition 6. *In a forward segment, the index of an unmatched left side vertex is larger than any matched left side vertex.*

Proof. Without loss of generality, consider the first forward segment. Let a_j be the unmatched left side vertex and let b_v be the last right side vertex in this segment. It suffices to show that the index of $mat[b_v]$ must be smaller than a_j. However, this is immediate due to Proposition 5. □

The following is the most important property of a forward segment. It enables us to add many vertices to the reachable set at the same time.

Proposition 7. *Suppose left side vertex a_s is matched to b_p. If a_s is in R, then in the same forward segment as a_s, left side vertices with no smaller indices than a_s and right side vertices with no smaller indices than b_p can all be added to R.*

Proof. Let b_v be the last right side vertex in this forward segment and suppose b_v is matched to a_t. Since the matching is noncrossing, left side vertices in the same forward segment with larger indices than a_s are either matched to $b_{p+1}, b_{p+2}, \ldots, b_v$, or unmatched (if there are any). By the definition of the forward segment, a possible alternating path is a_s to b_p, b_p to $mat[b_{p+1}]$, $mat[b_{p+1}]$ to $b_{p+1}, \ldots, b_{v-1}$ to a_t, a_t to b_v, then to any of the unmatched vertices. □

For example, in Fig. 5c, once the alternating path reaches a_4 in G_3, left side vertices with indices no smaller than a_4 and right side vertices with indices no smaller than b_4 can all be added to R. Note that this is a much efficient way to expand the reachable set since many vertices including

those not adjacent to a_4, are added to R in one single operation.

The following properties can also be easily verified:

Proposition 8. *Any nonisolated vertex is in exactly one forward segment.*

Proposition 9. *Nonisolated vertices on the same wavelength are in the same forward segment.*

We will denote the pth forward segment in G_i as FS_p^i. For a forward segment, the following information is needed in our algorithm: 1) Whether there is an unmatched left side vertex in this segment. 2) The wavelength indices of buffer vertices in the segment. The first item can be stored in a single bit. The second item is stored in a k-bit register, denoted as $fsreg_p^i$ for FS_p^i, in which bit l is "1" if there is a buffer vertex on λ_l in FS_p^i. For example, the information about the second forward segment in G_3 is {*yes*; "000010XX"}, which means that in this segment there is an unmatched left side vertex, and there are buffer vertices on λ_5. "X" means "don't care" since there cannot be buffer vertices on wavelengths higher than b_x assigned to any subgraph.

The *backward segment* is defined in a similar way, only the scanning direction is now reversed to backwards. Imagine starting at b_u where b_u is the last right side vertex in G_i, and scan back to $b_{u-1}, b_{u-2}, \ldots$, till b_w when b_w is not adjacent to $mat[b_{w-1}]$. b_w to b_u and all the left side vertices matched to them is called a *backward segment*. After finding the first backward segment, start scanning from b_{w-1} to find other backward segments. Note that unlike the forward segments which are numbered from top to bottom, the backward segments are numbered in the reverse direction, or from bottom to top, in accordance to the scanning direction. For example, in Fig. 5c, both G_1 and G_3 have only one backward segment. G_2 has two backward segments. The first one has vertex set a_8 to a_{10} and b_8 to b_{10}, and the second one has vertex set a_1 to a_7 and b_1 to b_7.

Proposition 10. *In a backward segment, let b_u and b_w be the right side vertices with the largest and the smallest index, respectively. Suppose b_u is matched to a_i and b_w is matched to a_j. Then, left side vertices between a_i and a_j are all matched.*

Proof. By contradiction. If this is not true, suppose a_l is not matched while $i > l > j$. Let a_s and a_t be matched left side vertices in this segment where $s < l$ and $t > l$. Furthermore, let a_s and a_t be such vertices that are closest to a_l. Note that, if a_t is matched to b_z, then a_s must be matched to b_{z-1} and b_z must be adjacent to a_s. By Property 1 of the request graph, b_z is also adjacent to a_l. But, since $l < t$, the First Available Algorithm would not have matched b_z to a_t. This is a contradiction. □

Similar to forward segments, it can be shown that

Proposition 11. *Suppose left side vertex a_s is matched to b_w. If a_s is in R, then in the same backward segment as a_s, left side vertices with no larger indices than a_s and right side vertices with no larger indices than b_w can all be added to R.*

Proposition 12. *Any matched vertex is in exactly one backward segment.*

Proposition 13. *Matched vertices on the same wavelength are in the same backward segment.*

We will denote the qth backward segment in G_i as BS_q^i. Note that unmatched vertices are not in any backward segment, and the information of a backward segment needed in our algorithm is only the wavelength indices of buffer vertices stored in register $bsreg_q^i$ for BS_q^i. For example, $bsreg_1^1$ is {"100000XX"} which means that there is a buffer vertex on λ_1.

3.4.4 Matching Buffer Vertices—Expanding the Reachable Set in Parallel

We are now ready to present the core of our algorithm, which is to expand the reachable set R in parallel. In the expanding process, we say a buffer vertex in R *discovers* a forward or backward segment if it is adjacent to some left side vertex in the segment. Left side vertices and right vertices satisfying conditions in Propositions 7 and 11 are also said to be discovered by this buffer vertex, and the buffer vertex is called the "discoverer" of these vertices. All the discovered vertices can be added to R; hence, in the rest of the paper, we will interchangeably say a vertex is "discovered" or a vertex is "added to R." Despite the technical details, the idea is quite simple. In short, the algorithm runs in "rounds." Before each round, the buffer vertices discovered in the previous round were made known to every subgraph. Then, *in parallel*, each subgraph uses these buffer vertices to discover more vertices in it. Each subgraph will use the buffer vertices to discover as many vertices as possible. If an unmatched left side vertices were discovered, the augmenting path is found. Otherwise, the reachable set needs to be further expanded, which can only be done by making use of the newly discovered buffer vertices to discover more vertices because at this time each subgraph has expanded the reachable set within itself to the maximum. Thus, the subgraphs will announce the newly discovered buffer vertices and then start the next round.

To elaborate, before the first round, b_x itself is added to R. Then, *in parallel*, each subgraph checks which vertices in itself that can be discovered by b_x. It can be shown that these vertices must be in the first backward segment of each subgraph. Note that there cannot be unmatched left side vertices since there is no unmatched vertex in backward segments. However, at this time, some buffer vertices may have been discovered. These buffer vertices can be used to discover more vertices that cannot be discovered by b_x. Each subgraph will announce the buffer vertices that were discovered in itself. The union of the buffer vertices announced by all N subgraphs will be known to all subgraphs. Then, the second round begins. In *parallel*, each subgraph will check which vertices can be discovered by the buffer vertices announced in the previous round. If unmatched left side vertices are discovered, the augmenting path is found. Otherwise, as before, each subgraph can announce the newly discovered buffer vertices. This process is repeated until an unmatched left side vertex is discovered or until after a round in which no new buffer vertex is discovered.

We can now explain how the augmenting path was found in the example in Fig. 5c. This is illustrated in Fig. 6.

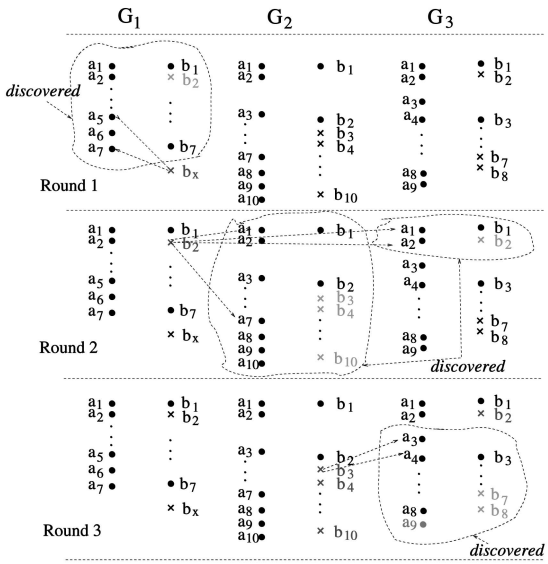


Fig. 6. Three rounds for finding the augmenting path for the example in Fig. 5c. For simplicity, the edges are not shown.

Round 1. Since b_x is on λ_6, it is only adjacent to a_5, a_6, and a_7 in G_1. It can discover the first backward segment in G_1, and vertices a_1 to a_7 and b_1 to b_7. Only one buffer vertex, b_2 in G_1, can be discovered. Thus, G_1 announces that there is a buffer vertex on λ_1. G_2 and G_3 do not make any announcement.

Round 2. Each of the subgraphs checks which vertices can be discovered by a buffer vertex on λ_1. G_1 finds out that these vertices have been discovered before. G_2 finds that the buffer vertex can discover the first forward and the second backward segment. G_3 finds that the buffer vertex can discover the first forward and the first backward segment. No unmatched left side vertices in these segments can be discovered; however, buffer vertices on λ_2, λ_3, and λ_4 in the first forward segment of G_2 can be discovered.

Round 3. G_3 finds that a buffer vertex on λ_2 can discover the second forward segment in it, and there is an unmatched left side vertex a_9 in this segment; thus, an augmenting path is found.

3.4.5 Matching Buffer Vertices—Details of Reachable Set Expanding

After explaining the general idea, in this section, we give the details of parallel reachable set expanding.

First, note that after a round, to announce the newly discovered buffer vertices, subgraph G_i needs only to write to a k-bit register, denoted as $breg_i$, in which bit i is "1" if a buffer vertex on λ_i in G_i has been discovered. The union of the N announcements can thus be obtained by doing an "or" operation to $breg_i$ for all $1 \leq i \leq N$, and suppose the result is stored in register $breg$. With $breg$, another register $lreg$ can be obtained which is the information needed in the next round. In $lreg$, bit i is "1" if left side vertices on λ_i can be discovered by the buffer vertices indicated by $breg$ and if these left side vertices have not been discovered before.

In the next round, each subgraph will examine $lreg$ by scanning through it. If subgraph G_i finds that bit l of $lreg$ is "1," it will check whether there are left side vertices on λ_l, and if yes, these left side vertices have been discovered. It can then find the forward and backward segment to which these left side vertices belong, and suppose they are FS_p^i and BS_q^i, respectively. From previous discussions, we know that left side vertices and right vertices in the same forward and backward satisfying conditions in Propositions 7 and 11 are also discovered. To be specific, left side vertices in FS_p^i on wavelengths no lower than λ_l and left side vertices in BS_q^i on wavelengths no higher than λ_l are discovered. If there are unmatched left side vertices in FS_p^i, the search is over. Otherwise, suppose a right side vertex on λ_h is matched to one of the left side vertices on λ_l. Then, right side vertices in FS_p^i on wavelengths no lower than λ_h and right side vertices in BS_q^i on wavelengths no higher than λ_h are discovered. We will refer to this right side vertex as a "pivot" vertex. Any right side vertex matched to left side vertex on λ_l can be a pivot vertex and, in practice, it can be the one with the smallest index. The subgraph needs to write a few "1"s to $breg_i$ according to the wavelengths of discovered buffer vertices. Let κ_h^f be a k-bit register where bit t is "1" only if $t \geq h$. Let κ_h^b be a k-bit register where bit t is "1" only if $t < h$. To update $breg_i$, we can "or" $breg_i$ with $fsreg_p^i \& \kappa_h^f$ and $bsreg_q^i \& \kappa_h^b$. Note that all these operations take *constant* time in all subgraphs; therefore, all subgraphs will finish checking $lreg$ at the same time in each round, which is the desired behavior of a parallel algorithm.

When an unmatched left side vertex is found, the algorithm needs to establish the M-augmenting path. Note that, if a vertex was discovered in a certain round, its discoverer must be discovered in the previous round, and there must be an M-alternating path from a vertex to its discoverer. When the unmatched left side vertex, say, a_t, is found, we will have a sequence of vertices, $a_t, b_u, b_v, b_w, \ldots, b_z, b_x$, where each vertex is discovered by the vertex next to it. In case a vertex has multiple discoverers, we can arbitrarily choose one. Therefore, we can start with a_t, and first find the M-alternating path to its discoverer b_u, then extend the M-alternating path from b_u to b_v, and so on, until the path extends to b_x.

However, in implementation, the augmenting path needs not be explicitly established. As mentioned earlier, to update the matching is actually to move some buffer vertices from one subgraph to another. We can first move b_u to the subgraph where a_t is in, then move b_v into the subgraph where b_u used to be in, then move b_w into the subgraph where b_v used to be in, ..., until b_x is moved into the subgraph where b_z used to be in. Then, the First Available Algorithm can be run in the subgraphs in parallel to obtain the new matching. All vertices involved in this moving should be able to be matched due to the properties of the forward and backward segments.

3.4.6 Matching Buffer Vertices—Proof for the Optimality of Reachable Set Expanding

We now prove the optimality of the algorithm.

Theorem 1. *When the algorithm terminates, it either discovers an unmatched left side vertex reachable from b_x or the entire reachable set of b_x.*

Proof. It suffices to show that if the algorithm terminates without reporting an augmenting path, it has discovered all the left side vertices reachable from b_x. We show this by contradiction. If this is not true, then there will be a left side vertex a_t, either matched or unmatched, that can be reached by an alternating path starting from b_x but was not discovered by the algorithm. Suppose in this hypothetical alternating path, a_t is proceeded by b_u, and b_u is proceeded by a_s. We claim that if a_t was not discovered, neither would a_s.

Suppose the claim is not true, that is, the algorithm has discovered a_s. Note that b_u must be matched to a_s thus was also discovered. If b_u is a buffer vertex, the algorithm would have discovered a_t since all vertices adjacent to buffer vertices were scanned. Thus, b_u must be an output vertex. Note that, if b_u is an output vertex, a_t must be in the same subgraph as a_s.

First, consider when $s < t$. In this case, if a_s and a_t are in the same forward segment, the algorithm would have discovered a_t; thus, a_s and a_t must be in different forward segments. However, if this is true, there must be an unmatched left side vertex between a_s and a_t since b_u is adjacent to a_t, by the definition of forward segments. Thus, the algorithm would have terminated by discovering an unmatched left side vertex, which is a contradiction. Thus, s cannot be smaller than t.

However, if $s > t$, according to Proposition 5 in Section 3.3, a_t must be matched. It is not hard to see that a_s and a_t must be in the same backward segment since b_u is adjacent to a_t. But, in this case, the algorithm must also have discovered a_t, which is also a contradiction. Therefore, the claim is true, i.e., the algorithm did not discover a_s.

Following the same argument, we can show that the algorithm also did not discover the left side vertex proceeding a_s in the alternating path, and this can be carried on until the conclusion that the algorithm did not discover any left side vertex in this alternating path. But, this contradicts with the fact that the algorithm at least discovers all the left side vertices directly adjacent to b_x and any alternating path starting from b_x must visit one of these vertices first. □

We summarize all the previous discussions and present the Parallel Segment Expanding Algorithm in Table 3 for finding optimal schedules in the WDM interconnect with shared buffer.

TABLE 3
Parallel Segment Expanding Algorithm

```
Run First Available Algorithm in all subgraphs in parallel
to cover maximum number of output vertices.
for every buffer vertex b_x do
  Try direct insertion in all subgraphs in parallel.
  If succeed, continue to the next buffer vertex.
  for i := 1 to N do in parallel
    if b_x can discover BS_1^i
      breg_i ← bsreg_1^i
    end if
  end for
  Generate lreg according to breg_i, i ∈ [1, N].
  while lreg is not all zero
    for i := 1 to N do in parallel
      Clear breg_i.
      for each '1' bit in lreg, say, bit l, do
        if no left side vertex on λ_l
          continue to next '1' bit;
        end if
        Suppose left side vertices on λ_l are in FS_p^i and BS_q^i.
        if there are unmatched left side vertices in FS_p
          break from the while loop;
        end if
        Update breg_i according to fsreg_p^i and bsreg_q^i.
      end for
    end for
    Generate lreg according to breg_i, i ∈ [1, N]. Reset a bit in
    lreg to '0' if it has been checked in previous rounds.
  end while
  Update the matching if an augmenting path is found
end for
```

4 Complexity Analysis and Performance Evaluation

4.1 Implementation Issues and Complexity Analysis

In this section, we give the complexity analysis of the algorithm. We show that by using N processing units or decision making units, the algorithm runs in $O(Bk^2)$ time. In addition, it should be mentioned that it is not difficult to implement the N processing units in hardware to reduce the cost and further speed up the scheduling process.

The key to reducing the running time and storage space is that we will work on *wavelengths* of vertices rather than the indices, because vertices on the same wavelength have the same adjacency sets and are in the same forward and backward segments. For each subgraph, we can store both left side and right side vertices in an array with k entries, denoted by $LS[]$ and $RS[]$, respectively, where each entry stores: 1) the number of vertices on this wavelength and 2) the index of forward and backward segments these vertices belong to. The forward and backward segments can be described by two intervals of integers because, in a segment, left side vertices and right side vertices are all consecutive. For example, the second forward segment in G_3 in Fig. 5 can be represented by $[3, 4]$ and $[2, 5]$, corresponding to left and right side vertices, respectively, which means that vertices with wavelengths in this range are in this segment. Each segment also needs a k-bit register to store the wavelengths of buffer vertices. In addition, the forward segment needs one bit to store the information of whether there is an unmatched left side vertex. The total storage space needed for all N subgraphs is $O(Nk(k + \log Nk))$ bits.

The first task, covering maximum number of output vertices, is to run First Available Algorithm on each subgraph with only output vertices. It can be done in parallel and takes $O(k)$ time. After this, the buffer vertices need to be matched one by one. We will show that to match a buffer vertex totally $O(k)$ time is needed. Suppose buffer vertex b_x is on wavelength λ_j. As mentioned earlier, first, direct insertion will be tried in parallel. For each subgraph, this is to increment the number of vertices of $RS[j]$ by one, then run the First Available Algorithm, which takes $O(k)$ time. If direct insertion fails, the augmenting path needs to be found. The

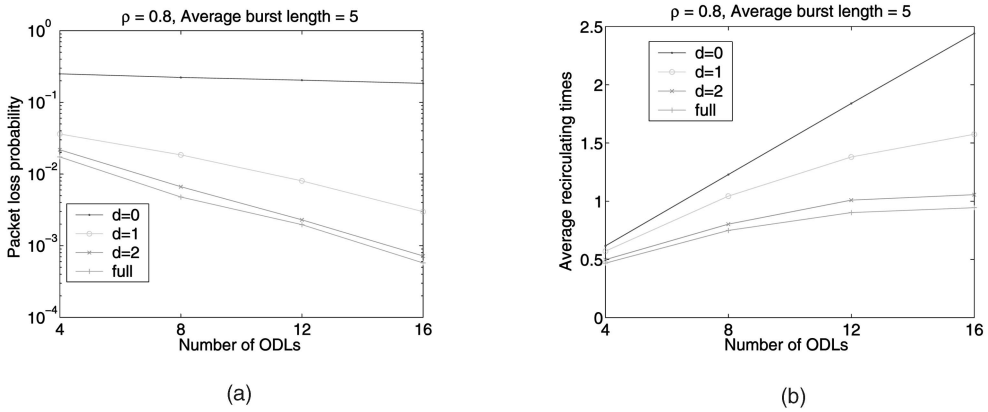


Fig. 7. Performance of a WDM interconnect with eight input/output fibers and eight wavelengths per fiber under bursty traffic. (a) Packet loss probability. (b) Average delay.

forward and backward segments in each subgraph can be found by a linear search. in $O(k)$ time. Then, at each round of reachable set expansion, each subgraph checks the content of register $lreg$. While scanning through $lreg$, simple hardware can be used to skip the "0" bits and scan only the "1" bits. As explained earlier, for each "1" bit in $lreg$, the work can be finished in constant time for all subgraphs. Also note that, in each round, the subgraph only checks the new wavelengths that have not been checked before. Thus, the total time spent on this task in the entire expansion process is $O(k)$ since there are k wavelengths.

Between two expansions, the content of $breg_i$ for $1 \leq i \leq N$ should be "OR"ed, and then be used to generate $lreg$. This function can be implemented in hardware which finishes this job in constant time. Also note that the number of expansion rounds can never exceed k, because after each round, there must be some "1" bit in $lreg$, i.e., there must be some newly discovered left side vertices. Therefore, the total time spent in generating $lreg$ is also $O(k)$.

If one unmatched left side vertex is found, the matching should be updated. As explained earlier, this is to move some buffer vertices from one subgraph to another which is to increment or decrement the corresponding entries in $RS[]$ of the subgraphs. Since the number of expansion rounds cannot exceed k, this will also take $O(k)$ time. Then, the new matching can be obtained in $O(k)$ time by running the First Available Algorithm on each subgraph.

Therefore, overall, it needs $O(k)$ time to match a buffer vertex. Thus, the total running time of the algorithm is $O(Bk^2)$ since there are Bk buffer vertices.

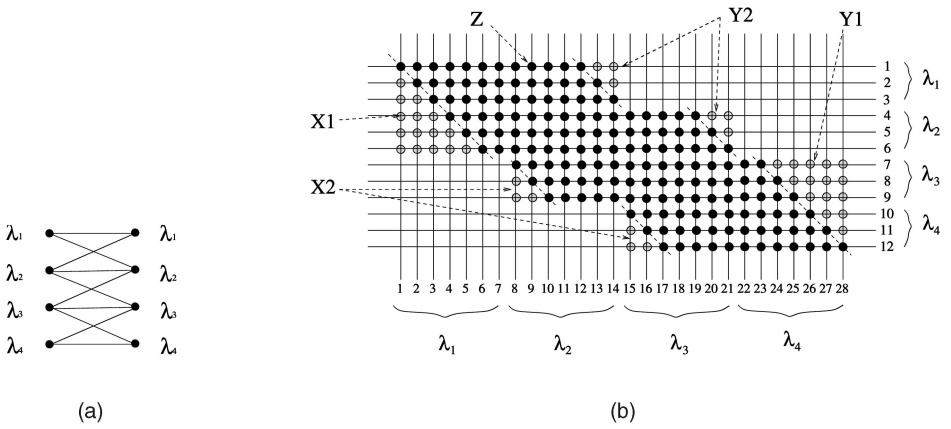
4.2 Performance Evaluation

We implemented Parallel Segment Expanding Algorithm in software and evaluated the performance of the WDM interconnect. The interconnect simulated has eight input fibers and eight output fibers with eight wavelengths on each fiber. We assume that the arrivals of the packets at the input channels are bursty: An input channel alternates between two states, the "busy" state and the "idle" state. When in the "busy" state, it continuously receives packets and all the packets go to the same destination. When in the "idle" state, it does not receive any packets. The length of the busy and idle periods follows geometric distribution. The durations of the packets are all one time slot and for each experiment the simulation program was run for 100,000 time slots.

Fig. 7a shows the packet loss probability (PLP) of the interconnect as a function of the number of fiber delay lines. The traffic load is 0.8, the average burst length and the average idle period are 5 and 1.25 time slots, respectively. Two wavelength conversion distances, $d = 1$ and $d = 2$ were tested, along with the no conversion case ($d = 0$) and the full conversion case. As expected, packet loss probability decreases as the number of delay lines increases. However, more significant improvements seems to come from the increase of the conversion ability. For example, when $d = 0$, the PLP curve is almost flat, which means that without wavelength conversion, there is scarcely any benefit by adding buffer to the interconnect. But, when $d = 1$, the PLP drops by a great amount as compared to $d = 0$, and the slope of the PLP curve is also much larger. We can also see that the PLP for $d = 2$ is already very close to the full wavelength conversion; therefore, there is no need to further increase the conversion ability. All this suggests that in a WDM interconnect, wavelength conversion is crucial, but the conversion needs not to be full range, and with a proper conversion distance, adding buffer will significantly improve the performance.

Fig. 7b shows the average delay of a packet as a function of the number of fiber delay lines. The average delay is actually the average rounds a packet needs to be recirculated before being sent to the output fiber. We can see that a larger conversion distance results in a shorter delay, and the delay for $d = 2$ is very close to the delay for full conversion. Fixing the conversion distance, the delay is longer for larger buffer sizes since when buffer is larger more packets are directed to buffers rather than being dropped.

We also compared the performance of the interconnect with another type of WDM interconnect studied in [6], [18], where the ODLs are not shared but dedicated to each output fiber. Calling the interconnect in this paper $\mathbf{I}_1$ and the interconnect to be compared with $\mathbf{I}_2$, we find that with the same conversion distance and the same number of ODLs, the PLP of $\mathbf{I}_1$ is much smaller than $\mathbf{I}_2$, though the delay time of $\mathbf{I}_1$ is slightly larger. For example, when $N = 8$, $k = 8$, $d = 2$, and when there are a total of 16 ODLs, the PLPs and the delay times for $\mathbf{I}_1$ versus $\mathbf{I}_2$ are 0.00072 versus 0.020 and 1.06 versus 0.4622, respectively. Therefore,



(a) (b)

Fig. 8. (a) Wavelength conversion for a four-wavelength interconnect. (b) Crosspoint layout for a switching fabric with $N = 4$, $B = 3$, $k = 4$, and the wavelength conversion as defined in Fig. 8a.

interconnect with shared buffer is a better choice for OPS networks.

5 Efficient Design of the Switching Fabric

In this section, we will study another important aspect of the WDM interconnects, which is the design of the switching fabric. The switching fabric is configured according to the decisions of the scheduling algorithms, and connects the inputs to the outputs. As mentioned earlier, to resolve the output contention buffers are added to the switch, however, this will result in a larger switching fabric. Hence, it is important to find efficient ways to build the switching fabric to reduce the cost of the interconnect.

In Fig. 1, the interconnect has one switching fabric with $(N + B)k$ inputs/outputs. However, we may not actually need to build such a switching fabric since, unlike an unbuffered packet which must be sent to the specified output fiber, a buffered packet can be sent to any of the ODLs. As a result, the switching fabric connecting the inputs to the delay lines needs not be so "powerful" as the switching fabric connecting the inputs to the output fibers. Hence, we can build two switching fabrics, one is $(N + B)k \times Nk$ which connects the inputs to the output fibers and the other is $(N + B)k \times Bk$ which connects the inputs to the delay lines. We will show that by adopting the idea of *concentration*, the cost of the latter, measured by the number of crosspoints, can be reduced by a significant amount.

In this section, we will assume the conversion distances for all wavelengths are the same, i.e., the wavelength conversion is "regular." However, this is only for presentational convenience and it is not difficult to generalize our designs to "irregular" wavelength conversion systems where different wavelengths may have different conversion distances.

5.1 Switching Fabric 1

The switching fabric we will design has $(N + B)k$ inputs and Bk outputs, because there are $N + B$ input fibers and B output fibers with respect to the switching fabric. The inputs and outputs can be ordered according to the wavelengths. To be specific, the $[(t - 1)(N + B) + p]$th input is wavelength channel λ_t on the pth input fiber and the $[(t - 1)B + p]$th output is wavelength channel λ_t on the pth output fiber. We use a_i where $1 \leq i \leq (N + B)k$ and b_u where $1 \leq u \leq Bk$ to denote the inputs and outputs.

With wavelength conversion ability, an input wavelength channel can be connected to output wavelength channels that can be converted from it. An example of the switching fabric is shown in Fig. 8b, where $N = 4$, $B = 3$, $k = 4$, and wavelength conversion is as defined in Fig. 8a. In this figure, each vertical line represents an input and each horizontal line represents an output. There is a node (either in black or in gray) at the intersection of input line a_i and output line b_u if there can be a connection between a_i and b_u, i.e., the wavelength of b_u can be converted from the wavelength of a_i. Apparently, if the wavelength conversion is full range there will be a node at every intersection. A possible design of the switching fabric is to let each node in Fig. 8b be a crosspoint, which will be referred to as "Switching Fabric 1."

For Switching Fabric 1 as well as for any switching fabric, given a set of inputs which are the set of packets that should be buffered, decided by the scheduling algorithm, there should be a method to assign the outputs to the inputs. We call the set of outputs (inputs) that can be connected to an input (output) the *adjacency set* of this input (output). Under Switching Fabric 1, the adjacency set of the inputs/outputs has the same properties as the vertices in the request graph in Section 3.3. By regarding the inputs as left side vertices and the outputs as right side vertices, it is not difficult to see that the First Available Algorithm can be used to find the maximum number of inputs that can be simultaneously connected to the outputs. That is, we can scan the inputs according to their indices, and assign an output b_u to input a_i if b_u is the output adjacent to a_i with the smallest index and is not assigned to any input yet.

We now define the *feasible input*.

Definition 1. *A group of inputs is* feasible *to a switching fabric if each of the inputs can be assigned to some output by the switching fabric.*

For Switching Fabric 1, a group of inputs is feasible if and only if each input in the group can be assigned to an output by the First Available Algorithm.

Switching Fabric 1 is the most powerful switching fabric one can have, given the wavelength conversion ability.

However, it is not optimal for our purpose, since many crosspoints are redundant. We will give a better design next.

5.2 Concentrators

We now show how to adopt the idea of concentration to reduce the switching cost. A $p \times q$ *concentrator* is a switching fabric with p inputs and q outputs where $p > q$, and for any q inputs, it is capable of assigning an output for each of the inputs [13], [14]. However, it does not guarantee that an input can be connected to a specific output. The concentrator only guarantees that there exists some output that this input can be connected to.

The switching fabric we are designing can also be a concentrator since an input packet needs only to be assigned to some wavelength channel and this wavelength channel can be on any delay line. However, because of the wavelength conversion constraint, the "concentrator" here is more complex than an electronic concentrator. We define a WDM concentrator with limited range wavelength conversion as follows:

Definition 2. *A WDM switching fabric with limited range wavelength conversion is a concentrator if any feasible input to Switching Fabric 1 is also feasible to it.*

In plain words, for anything Switching Fabric 1 can do, so does the concentrator. Any set of packets that are to be buffered, decided by the Parallel Segment Expanding Algorithm or any other correct scheduling algorithms, must be feasible to Switching Fabric 1. If so, they must also be feasible to a concentrator, i.e., for every packet there must be a unique wavelength channel on the ODLs that can be connected to.

5.3 Switching Fabric 2

We now divide the crosspoints of Switching Fabric 1 into three areas, *area X*, *area Y*, and *area Z*, as shown in Fig. 8b, and we will show that a concentrator needs only the crosspoints in area Z.

From the figure, we can see that area X contains the triangle areas with cyan nodes in the lower half of the switching matrix; area Y contains the triangle areas with cyan nodes in the upper half of the switching matrix, symmetrical to area X; and area Z is the rest of the switching matrix. More formal definitions of these areas are given as follows.

There are two subareas of area X, X1, and X2, where X1 is a single large triangle area and X2 is a group of small triangle areas. For area X1, the easiest way to understand it is to imagine drawing a 45 degree line starting from (a_1, b_1), and extending to $(a_2, b_2), (a_3, b_3), \ldots$, until (a_I, b_I) when node (a_{I+1}, b_{I+1}) does not exist. Nodes under this line are said to be in area X1. In Fig. 8b, $I = 6$, i.e., the line stops at node (a_6, b_6). Note that, if node (a_i, b_u) is in this area, then $u > i$, and input a_i is adjacent to output b_w for all $w \leq I$.

Next, we define area X2. If the imaginary 45 degree line described above stops at (a_I, b_I), either b_{I+1} does not exist or b_I is the end of the adjacency interval of a_{I+1}. In the first case, $I = kB$ and there is no area X2. Otherwise, there is area X2, and two parameters, t and h, are involved for defining it, where t is the wavelength of output b_I and h is the wavelength of input a_I. In the example, $t = 2$ and $h = 1$. For the B outputs on λ_{t+1}: b_{tB+1} to b_{tB+B}, the first (with the smallest index) B inputs adjacent to them are $a_{h(N+B)+1}$ to $a_{h(N+B)+B}$. Imagine drawing a 45 degree line crossing nodes $(a_{h(N+B)+1}, b_{tB+1})$, $(a_{h(N+B)+2}, b_{tB+2}), \ldots, (a_{h(N+B)+B}, b_{tB+B})$. Nodes under this line are said to be in area X2 for output wavelength λ_{t+1}. In Fig. 8b, this line crosses (a_8, b_7), (a_9, b_8), and (a_{10}, b_9) and nodes in area X2 for output wavelength λ_3 are (a_8, b_8), (a_8, b_9), and (a_9, b_9). For any node (a_j, b_v) in this area, we have $j - h(N + B) < v - tB$, and a_j is adjacent to output b_w for all $tB + 1 \leq w \leq tB + B$. Area X2 for output wavelengths $\lambda_{t+2}, \lambda_{t+3}, \ldots, \lambda_k$ can be defined in a similar way.

The definitions for areas Y1 and Y2 are similar to areas X1 and X2, except that the imaginary 45 degree line starts from the other end (node $(a_{(N+B)k}, b_{Bk})$) and the nodes *above* the line are in area Y. Nodes in areas Y1 and Y2 also have similar properties as nodes in areas X1 and X2.

If the First Available Algorithm is run on Switching Fabric 1, we have:

Lemma 1. *The First Available Algorithm does not use the crosspoints in area X.*

Proof. Consider area X1 first. If node (a_i, b_u) is in area X1 and a_i is assigned to b_u by the First Available Algorithm, b_1 to b_{u-1} cannot be available to a_i when a_i is checked. This only occurs when all these outputs are assigned to some inputs with smaller indices than a_i, since a_i is adjacent to all these outputs. But, this is impossible since there are at most $i - 1$ inputs before a_i and $u > i$.

The proof for the first block of area X2, i.e., the area for λ_{t+1} is similar. Suppose node (a_j, b_v) is in area $X2$ and a_j was assigned to b_v by the algorithm. Note that, inputs from a_1 to $a_{h(N+B)}$ can only be assigned to b_1 to b_{tB}, and a_j is adjacent to all b_{tB+1} to b_{v-1}. Thus, if a_j is assigned to b_v, b_{tB+1} to b_{v-1} must be all assigned to inputs from $a_{h(N+B)+1}$ to a_{j-1}. This contradicts with the fact that $j - h(N + B) < v - tB$. Similarly, we can prove for other blocks of area $X2$. □

Lemma 2. *After running the First Available Algorithm on Switching Fabric 1, any assignment in area Y can be replaced with an assignment in area Z without causing any collision.*

Proof. Consider area Y1. If there are assignments in this area, let a_i be the one with the largest index and suppose it is assigned to b_u. By the algorithm, outputs from b_{u+1} to b_{kB} are not assigned to any inputs. Hence, we can assign a_i to b_{kB} and, after that, b_u to b_{kB-1} will be free and we can assign the active input with the second largest index, say, a_j, to b_{kB-1}, and b_v to b_{B-2} will be free, where b_v was used to be assigned to a_j. This reassignment can be carried on, and in the pth reassignment the active input with the pth largest index will be assigned to b_{kB-p+1}. Note that, after the reassignment, the input will be assigned to an output with a larger index than the output it used to be assigned to, therefore, not colliding with the assignments for inputs with smaller indices, and all new assignments are in area Z.

Following similar arguments, we can show that the assignments in area Y2 can be replaced with nodes in area Z and the new assignments will also not collide with other existing assignments. □

Calling the switching fabric with crosspoints only in area Z "Switching Fabric 2," clearly, we have:

Theorem 2. *Any feasible inputs to Switching Fabric 1 are also feasible to Switching Fabric 2. As a result, Switching Fabric 2 is a concentrator.*

An efficient algorithm is needed to assign outputs to inputs in Switching Fabric 2. Luckily, we can simply use the First Available Algorithm, since the adjacency set of inputs/outputs in Switching Fabric 2 also has the properties in Section 3.3. Practically, since there are fewer outputs than inputs, we can run the First Available Algorithm on the output side and the time complexity is $O(Bk)$.

The following theorem indicates that Switching Fabric 2 uses the minimum number of crosspoints for this type of crosspoint layout.

Theorem 3. *Switching Fabric 2 will not be a concentrator if any crosspoint is removed from it.*

Proof. We prove this by showing that, if any one of the crosspoints is removed, the remaining switching fabric will fail to satisfy a set of feasible inputs. For notational convenience, we define "area X inputs." Input a_j is an area X input if the 45 degree lines used in the definition for area X crosses some node (a_j, b_v). In the example of Fig. 8b, they are inputs 1-6, 8-10, and 15-17. Similarly, define "area Y inputs" as the inputs involved in the 45 degree lines for the definition of area Y. In the example they are inputs 12-14, 19-21, and 23-28.

Now, suppose crosspoint (a_i, b_u) was removed and suppose when defining area Y, the 45 degree line crosses node (a_j, b_u). Consider the set of inputs consisting of all area X inputs with smaller indices than a_i and all area Y inputs with larger indices than a_j, plus input a_i. For example, in Fig. 8b, if crosspoint (a_6, b_5) was removed, a_j is a_{20}, and the set of inputs will be a_1 to a_4, a_{21}, a_{23} to a_{28}, plus a_6. Note that a_1 is only adjacent b_1 and, therefore, must be assigned to b_1. Given this, a_2 though adjacent to b_1 and b_2, must also be assigned to b_2. This argument is carried on, and as a result, outputs except for b_u must be assigned to inputs except for a_i. In the example, b_1 to b_4 must be assigned to a_1 to a_4, b_6 must be assigned to a_{21}, and b_7 to b_{12} must be assigned to a_{23} to a_{28}. Therefore, no output can be assigned to a_i. But, the set of inputs would be feasible had (a_i, b_u) not been removed. □

5.4 Cost Comparison

In this section, we will see how much this concentrator design reduces the switching fabric cost in terms of the number of crosspoints.

Let the conversion distance be d and note that the number of nodes in area X as well as in area Y is

$$\alpha = kB(B-1)/2 + B^2t(t-1)/2, \tag{1}$$

where t is the largest integer that satisfies

$$\left\lceil \frac{Bt}{N+B} \right\rceil + d \geq t. \tag{2}$$

The number of crosspoints in Switching Fabric 1 is

$$\beta = B(N+B)[k + 2kd - d^2 - d], \tag{3}$$

assuming $k > 2d$. The saved cost is $2\alpha/\beta$. Fig. 9 shows the saved cost by using Switching Fabric 2 when $N = 16$ and $k = 16$ for different B and d. We can see that in general the concentrator design will save more when B is large, and at least about 10 percent of the cost can be saved.

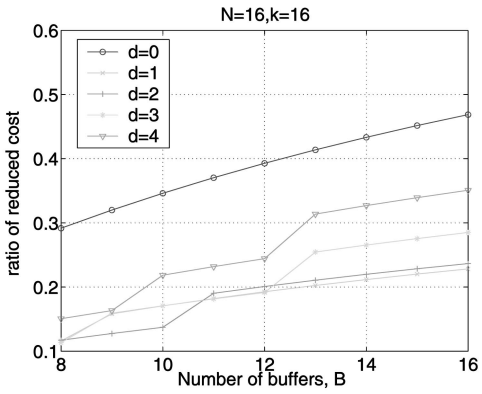


Fig. 9. Cost saved by using the concentrator design.

6 Conclusions

In this paper, we have studied WDM optical interconnects with limited range wavelength conversion and shared buffer. We first considered the scheduling problems in this type of interconnect and gave an optimal parallel algorithm called the Parallel Segment Expanding Algorithm that finds a schedule that both maximizes throughput and minimizes packet delay. The proposed algorithm runs in $O(Bk^2)$ time, as compared to $O((N+B)^3k^3)$ time if directly applying other existing algorithms, where N is the number of input/output fibers, B is the number of optical delay lines (ODL), and k is the number of wavelengths per fiber. We also considered the switching fabric design for this type of interconnect. We distinguished between the switching fabric connecting the inputs to the output fibers and the switching fabric connecting the inputs to the ODLs, and gave a new design based on the idea of concentration which significantly reduces the cost of the latter in terms of the number of crosspoints. Our future work includes analytical performance evaluation of the interconnect under different traffic models.

Acknowledgments

This research was supported in part by the US National Science Foundation under grant numbers CCR-0207999 and ECS-0427345.

References

[1] B. Mukherjee, "WDM Optical Communication Networks: Progress and Challenges," *IEEE J. Selected Areas in Comm.*, vol. 18, no. 10, pp. 1810-1824, 2000.

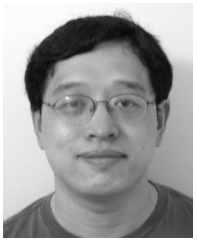
[2] D.K. Hunter, M.C. Chia, and I. Andonovic, "Buffering in Optical Packet Switches," *J. Lightwave Technology*, vol. 16, no. 12, pp. 2081-2094, 1998.

[3] M. Kovacevic and A. Acampora, "Benefits of Wavelength Translation in All-Optical Clear-Channel Networks," *IEEE J. Selected Areas in Comm.*, vol. 14, no. 5, pp. 868-880, 1996.

[4] S.L. Danielsen et al., "Analysis of a WDM Packet Switch with Improved Performance Under Bursty Traffic Conditions Due to Tunable Wavelength Converters," *J. Lightwave Technology*, vol. 16, no. 5, pp. 729-735, 1998.

[5] T. Tripathi and K.N. Sivarajan, "Computing Approximate Blocking Probabilities in Wavelength Routed All-Optical Networks with Limited-Range Wavelength Conversion," *IEEE J. Selected Areas in Comm.*, vol. 18, pp. 2123-2129, 2000.

[6] G. Shen et al., "Performance Study on a WDM Packet Switch with Limited-Range Wavelength Converters," *IEEE Comm. Letters,* vol. 5, no. 10, pp. 432-434, 2001.
[7] R. Ramaswami and K.N. Sivarajan, *Optical Networks: A Practical Perspective.* Morgan Kaufmann, 1998.
[8] L. Xu, H.G. Perros, and G. Rouskas, "Techniques for Optical Packet Switching and Optical Burst Switching," *IEEE Comm. Magazine,* pp. 136-142, 2001.
[9] R. Ramaswami and G. Sasaki, "Multiwavelength Optical Networks with Limited Wavelength Conversion," *IEEE/ACM Trans. Networking,* vol. 6, pp. 744-754, 1998.
[10] X. Qin and Y. Yang, "Nonblocking WDM Switching Networks with Full and Limited Wavelength Conversion," *IEEE Trans. Comm.,* vol. 50, no. 12, pp. 2032-2041, 2002.
[11] Y. Yang, J. Wang, and C. Qiao, "Nonblocking WDM Multicast Switching Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 12, pp. 1274-1287, Dec. 2000.
[12] S.L. Danielsen et al., "WDM Packet Switch Architectures and Analysis of the Influence of Tunable Wavelength Converters on the Performance," *J. Lightwave Technology,* vol. 15, no. 2, pp. 219-227, 1998.
[13] S. Nakamura and G.M. Masson, "Lower Bounds on Crosspoint in Concentrators," *IEEE Trans. Computers,* pp. 1173-1178, 1982.
[14] A. YavuzOruc and H.M. Huang, "Crosspoint Complexity of Sparse Crossbar Concentrators," *IEEE Trans. Information Theory,* pp. 1466-1471, 1996.
[15] N. McKeown, "The iSLIP Scheduling Algorithm Input-Queued Switch," *IEEE/ACM Trans. Networking,* vol. 7, pp. 188-201, 1999.
[16] M. Karpinski and W. Rytter, *Fast Parallel Algorithms for Graph Matching Problems.* Oxford Univ. Press, 1998.
[17] V. Eramo, M. Listanti, and M. DiDonato, "Performance Evaluation of a Bufferless Optical Packet Switch with Limited-Range Wavelength Converters," *IEEE Photonics Technology Letters,* vol. 16, no. 2, pp. 644-646, 2004.
[18] Z. Zhang and Y. Yang, "Scheduling in Buffered WDM Packet Switching Networks with Arbitrary Wavelength Conversion Capability," *Proc. IEEE INFOCOM Conf.,* 2004.
[19] W.J. Goralski, *Optical Networking and WDM,* first ed. McGraw-Hill, 2001.
[20] H. Qin, S. Zhang, and Z. Liu, "Dynamic Routing and Wavelength Assignment for Limited-Range Wavelength Conversion," *IEEE Comm. Letters,* vol. 5, no. 3, pp. 136-138, 2003.
[21] X. Masip-Bruin et al., "Routing and Wavelength Assignment under Inaccurate Routing Information in Networks with Sparse And Limited Wavelength Conversion," *Proc. IEEE GLOBECOM Conf.,* vol. 5, pp. 2575-2579, 2003.
[22] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston, 1976.
[23] W. LipskiJr and F.P. Preparata, "Algorithms for Maximum Matchings in Bipartite Graphs," *Naval Research Logistics Quarterly,* vol. 14, pp. 313-316, 1981.
[24] D.K. Hunter and I. Andronovic, "Approaches to Optical Internet Packet Switching," *IEEE Comm. Magazine,* vol. 38, no. 9, pp. 116-122, 2000.
[25] C. Develder, M. Pickavet, and P. Demeester, "Assessment of Packet Loss for an Optical Packet Router with Recirculating Buffer," *Proc. Conf. Optical Network Design and Modeling,* pp. 247-261, 2002.
[26] Z. Zhang and Y. Yang, "A New Switching Fabric Design for WDM Packet Switching Networks with Wavelength Conversion and Recirculating Buffering," *Proc. IEEE Int'l Conf. Comm.,* 2004.
[27] Z. Zhang and Y. Yang, "Optimal Scheduling in WDM Optical Interconnects with Arbitrary Wavelength Conversion Capability," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 11, pp. 1012-1026, Nov. 2004.
[28] D.B. West, *Introduction to Graph Theory.* Prentice-Hall, 1996.



Zhenghao Zhang received the BEng and MS degrees in electrical engineering from Zhejiang University, China, in 1996 and 1999, respectively. From 1999 to 2001, he worked in industry as a software engineer for embedded systems design. Since 2001, he has been studying toward the PhD degree in the Department of Electrical and Computer Engineering at the State University of New York at Stony Brook. His research interest includes scheduling and performance analysis of optical and wireless networks. He is a student member of the IEEE.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a professor of computer engineering and computer science at the State University of New York at Stony Brook. Her research interests include parallel and distributed computing and systems, high speed networks, optical and wireless networks, and high-performance computer architecture. Her research has been supported by the US National Science Foundation (NSF) and US Army Research Office (ARO). Dr. Yang has published extensively in major journals and refereed conference proceedings and holds six US patents in these areas. She is an editor for the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Parallel and Distributed Computing*. She has served on NSF review panels and program/organizing committees of numerous international conferences in her areas of research. She is a senior member of the IEEE and the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.