A High Speed Hardware Scheduler for 1000-port Optical Packet Switches to Enable Scalable Data Centers

Joshua L.Benjamin, Adam Funnell, Philip M.Watts
University College London
London, United Kingdom,
Email: joshua.benjamin.09@ucl.ac.uk

Benn Thomsen
Microsoft Research
Cambridge, United Kingdom

***Abstract*— Meeting the exponential increase in the global demand for bandwidth has become a major concern for today's data centers. The scalability of any data center is defined by the maximum capacity and port count of the switching devices it employs, limited by total pin bandwidth on current electronic switch ASICs. Optical switches can provide higher capacity and port counts, and hence, can be used to transform data center scalability. We have recently demonstrated a 1000-port star-coupler based wavelength division multiplexed (WDM) and time division multiplexed (TDM) optical switch architecture offering a bandwidth of 32 Tbit/s with the use of fast wavelength-tunable transmitters and high-sensitivity coherent receivers. However, the major challenge in deploying such an optical switch to replace current electronic switches lies in designing and implementing a scalable scheduler capable of operating on packet timescales.**

In this paper, we present a pipelined and highly parallel electronic scheduler that configures the high-radix (1000-port) optical packet switch. The scheduler can process requests from 1000 nodes and allocate timeslots across 320 wavelength channels and 4000 wavelength-tunable transceivers within a time constraint of 1µs. Using the Opencell NanGate 45nm standard cell library, we show that the complete 1000-port parallel scheduler algorithm occupies a circuit area of 52.7mm^2, 4-8x smaller than that of a high-performance switch ASIC, with a clock period of less than 8ns, enabling 138 scheduling iterations to be performed in 1µs. The performance of the scheduling algorithm is evaluated in comparison to maximal matching from graph theory and conventional software-based wavelength allocation heuristics. The parallel hardware scheduler is shown to achieve similar matching performance and network throughput while being orders of magnitude faster.

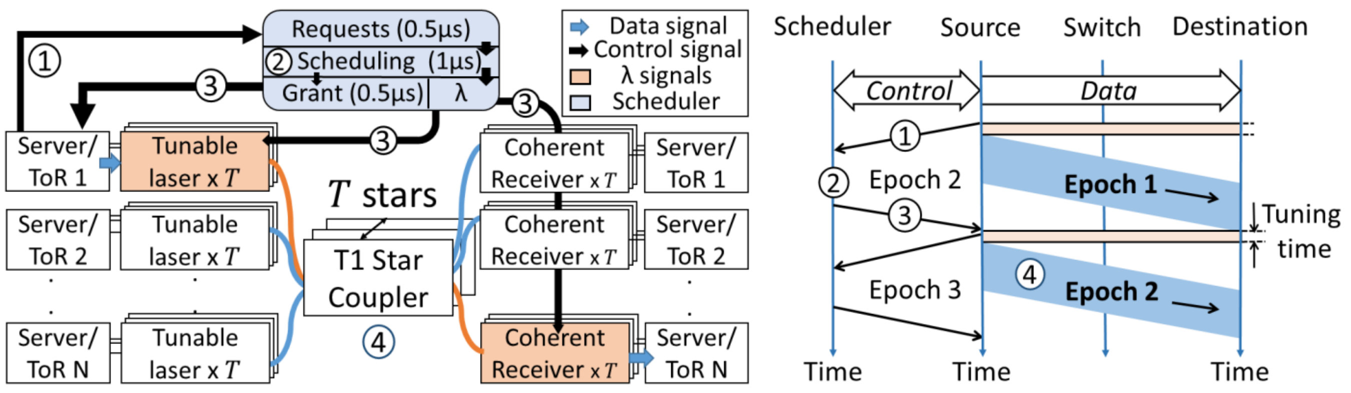
Keywords — scalability; data center; optical switch; star coupler; coherent receiver; scheduler; hardware algorithm; ASIC synthesis; performance analysis;

I. INTRODUCTION

Over the last two decades, there has been an aggressive increase in the rate of data being generated and shared around the world leading to increasing intra-datacenter traffic. Cisco's Global Cloud Index (GCI) predicts that the annual global data center IP traffic will increase from 0.39 ZB per month in 2015 to 1.30 zettabyes (ZB) per month by 2020 [1]. In order to support this data growth rate, data centers have been forced to scale by increasing the number of servers they support or by restructuring their interconnect architecture. The scalability of any data center architecture is defined by the maximum capacity and port count of the switching devices it employs. Data centers today use electronic switch ASICs which are limited in capacity by the product of pin bandwidth and the number of high-speed signal pins [2], which is only growing at a slow rate. For example, 256 x 25Gb/s lanes enables Tomahawk II to handle up to 6.4 Tb/s [3], however such ASICs have only 64 ports when bandwidth per port scales to 100 Gb/s. Creating a large data center out of low-radix electronic switches requires a very large number of ASICs and transceivers in a multi-level network hierarchy which is costly and difficult to manage. In addition, unused bandwidth is trapped within specific links even in oversubscribed scenarios. Optical switches can decouple the relationship between capacity and port count by building a pool of bandwidth, which can be shared by an increased number of nodes. The result is an optical switch which can scale to high port counts while maintaining high bandwidth per port. The fundamental limitations in scaling optical switches to high radix are optical signal loss and scheduling complexity. The benefits of employing high-radix optical switching devices in data center networks have been previously identified [4]. However, to avoid scheduling complexity, complexity has instead been introduced to the optical switching plane. Some examples include the use of tunable wavelength converters or multi-stage optical architectures [5]–[7]. Multi-stage architectures with distributed scheduling have also been demonstrated [8].

Recently, we proposed an optical switch that can scale to over 1000-ports in the data plane with the use of a passive star coupler core, tunable DS-DBR transceivers and high-sensitivity coherent receivers [9]. The experiments demonstrated 25 Gb/s per port with a simplified low cost and low power receiver, but could scale to 100+ Gb/s per port by using a full coherent receiver and Digital Signal Processing (DSP). However, such a switch requires a central scheduler to avoid contentions in its passive core. Previous high port-count scheduling work has used a software defined networking (SDN) approach to compute routing, wavelength and timeslot allocation (RWTA) for TCP/IP flows, however, the computation takes 53ms [10] making it unsuitable for scheduling at the packet level. In this work, we propose a

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/I004157/2 and by a PhD studentship awarded to Joshua Benjamin by the EPSRC-funded UCL-Cambridge Centre for Doctoral Training in Integrated Photonic and Electronic Systems (IPES).



(a) Central scheduler controlling the optical switch (b) Dataflow: Data and Control Plane

Figure 1: 1000-port Optical Packet Switch Architecture: Control and Data Plane

parallel iterative scheduler algorithm for the 1000-port optical switch [9] that can compute a schedule on packet timescales. We evaluate the scheduler scalability, demonstrating that it is viable to implement on 45nm CMOS ASIC and can compute a 1000-port schedule in 1 μs. Secondly, we assess the matching performance of the parallel hardware algorithm compared with conventional software based wavelength assignment techniques and maximal matching (graph theory). Thirdly, we evaluate the latency of the scheduling technique over multiple epochs.

This paper is organized as follows: Section II describes the 1000-port optical packet switch on which the scheduler design is based. Section III describes the parallel hardware scheduler algorithm as well as alternative scheduling algorithms against which it will be compared. Section IV presents details of the implementation of the hardware scheduler, providing evidence that it is viable to fabrication as a CMOS ASIC and will meet aggressive timing and throughput requirements. Section V compares the matching and latency performance of the parallel hardware scheduler against maximal matching and serial wavelength allocation heuristics, before section VI gives the overall conclusions.

II. Assumed Optical Packet Switch Architecture

A. *Optical Packet Switch: Data Plane*

The proposed switch architecture employs a passive star-coupler core, tunable DS-DBR lasers for transmission and coherent receivers with independently tunable local oscillators for their high sensitivity and fast wavelength selectivity. As shown in Figure 1(a), each node has up to T tunable lasers and T coherent receivers, where T is the total number of star couplers used. The switch scalability is limited by the splitting loss of each passive star coupler. Using the high sensitivity coherent receiver, a device now widely used in long-haul optical communication, it was shown that a single star coupler can scale to N=1000 ports; using T star couplers in parallel increases the available connectivity between nodes.

On each star coupler, choosing wavelengths within the optical C-band and on 50 GHz ITU grid spacing allows the use of 80 wavelength channels. Each star coupler connects all 1000 nodes, multiplexing and broadcasting W = 80 wavelengths to all receivers, where a single wavelength is selected to receive data. Since the number of nodes N is much greater than the number of wavelengths W, each wavelength can be shared between multiple transmitters, interleaving data from multiple nodes in a time-slotted manner (Time Division Multiplexing, or TDM). The work in [11] demonstrated a low cost simplified coherent receiver. However, the total capacity of an N = 1000 node optical switch can be as high as 32Tb/s (without accounting for tuning time), when using T=4 transceivers per node, W=80 wavelengths and DP-QPSK modulation transmission at 25Gbaud/s (a line rate of 100Gb/s).

To reconfigure the switch, transceivers are tuned every 2μs, to enable new connections to be made across the switch. Useful communication cannot take place while the laser is tuning to a new wavelength which was shown to take a maximum time of 200ns [11]. This results in a laser tuning overhead of less than 10% of the available time, during which data transmission is not feasible on the data plane. Figure 1(b) shows the useful data transmission time (called here an *epoch*) and the laser tuning time of the switch data plane.

As mentioned above, TDM is used on the data plane to reduce the overall switch latency by allowing many more transmitters to access the network in a given epoch than if the switch was routed by wavelength alone. Our previous experimental demonstration showed that up to M = 25 transmitters can share a wavelength in a given epoch [9]. In this switch prototype, there are L=100 timeslots, so each timeslot is 20 ns long, which at 100 Gb/s, corresponds to 250 bytes. Data center traffic is bimodal and hence, the nodes must be able to send several mice flow packets (few timeslots) and elephant flow packets (several adjoining timeslots). The node specifies the number of slots it needs as part of its request to the scheduler.

B. *Optical Packet Switch: Control Plane*

Figure 1(b) shows the working of the data and control plane in harmony. The nodes send wavelength and timeslot resource requests to the scheduler during one epoch to perform allocation for the subsequent epoch, shown as (1) in figure 1(a). The scheduler processes the requests and computes TDM slot grants for the nodes and laser and receiver wavelength assignments (2). The generated grants and transceiver wavelength assignments are then communicated back to the nodes (3). Finally, the optical switch is configured and the required data is communicated (4). Hence, the control plane has 2μs (the duration of an epoch) to collect requests, compute schedule and send grants. The control plane can send up to 6250 bytes of requests in 0.5μs, when using one wavelength channel at 100 Gb/s. Hence, allowing a total transit time for both requests and grants of 1μs, the scheduler has 1μs to complete the computation of the schedule for a single epoch. A practical scheduler can only accept a fixed number of requests, R, from each node, constrained by the control plane bandwidth and scheduler memory.

III. Scheduler Algorithms

This section describes the three scheduling algorithms examined for the switch architecture described in Section II. We use maximal matching as the ideal case and serial wavelength assignment heuristics as typical of a software implementation. Finally, we describe our parallel scheduling technique which is optimized to meet the timing requirements.

A. Maximal Matching (A1)

The first algorithm is maximal matching, based on graph theory. The request or the demand matrix creates a bipartite graph between a set of source nodes and a set of destination nodes that has N*R edges. The request also specifies the size or the number of slots it requires; this is a weighted bipartite maximal matching problem [12]. Maximal matching, in itself, is a P-type problem but the restrictions on the number of wavelength channels, timeslots and stars (edges) make the resource mapping an NP-hard problem [13]. The maximal matching discussed in this paper, however, aims to achieve maximal matching per iteration, allowing the allocation of W=80 channels to maximal flows. At each iteration, nodes requesting the highest number of slots are considered; up to W dominant flows are identified and allocated different wavelengths. Once maximal matching is achieved on one star, the algorithm moves to consider requests for the resources on the consequent star. The purpose of this algorithm is to help us perceive how large a matching one can achieve and it serves as the ideal case.

The maximal matching algorithm uses sorting elements and search functions to achieve its goal. To implement this technique in hardware for 1000 ports, search and sorting algorithms must be used at each iteration. The complexity of these algorithms makes them impractical for hardware implementation.

B. Serial Wavelength Assignment Heuristics (A2)

The second set of algorithms is derived from conventional wavelength assignment heuristics which loop through every request in a serial fashion [14] and would typically be implemented in software although a serial hardware implementation is also possible. Considering a single node-pair at a time, the software scans across all available wavelengths and stars to schedule.

Three software-based heuristics, random, least used and least loaded, are adapted for the optical switch described in section II and differ in the way that they choose wavelengths and stars for each assignment. The random wavelength assignment scheme chooses a random available wavelength on a random star. The least used wavelength assignment scheme, conscious of the limits of the number of transmitters that can share a wavelength in any epoch, M (=25), searches across all stars for the wavelength with minimum number of transceivers tuned to it. The least loaded wavelength assignment scheme is similar to the least loaded technique but it searches across all stars for the wavelength with minimum number of slots used. The pseudo-code for the software algorithms is described below:

Software Algorithms – *Serial Case*

```
Require: Set of nodes N, set of wavelengths W, demand matrix D, number of
stars T, slots per epoch L, maximum transceivers per wavelength M, Number
of iterations SerI. Circular queue of node pairs with non-zero demand – Q<i,j>
Ensure: Assign timeslots across wavelengths and stars to obtain (a) Random,
(b) Least loaded (LL) or (c) Least used (LU) assignment subject to constraints.
1:   $m_w = 0$, $slot[w] = 0 \ \forall w \in W$ {$m_w$ is the number of transceivers assigned
     to wavelength, slot[w] is the next available timeslot in wavelength w}
2:   $\lambda_{full}[w] = 0$ {wavelength availability: full = 1, available = 0}
3:   $Tx[n, t] = 0$, $Rx[n, t] = 0 \ \forall t, \tau \in T$ {where Tx is the wavelength assigned
     to node n transmitter and Rx is the wavelength assigned to node n
     receiver}
4:   for i = 1 to |SerI|
5:       /* Only one of the following three algorithms is used at a given time: */
6a:      $[\lambda, star]$ = random($\sim\lambda_{full}[w]$,T) {Random w in random star T}
6b:      $[\lambda, star]$ = leastloaded($\sim\lambda_{full}[w]$,T) {Find w using min slots}
6c:      $[\lambda, star]$ = leastused($\sim\lambda_{full}[w]$,T) {Find w using min trasceivers}
7:       /* Check for common wavelength first across all stars*/
8:       if ($Tx[i,t]==Rx[j,t]$ && $Tx[i,t]\neq 0$ && $|\lambda_{full}[i,w]| \neq 0$)
9:           Grant = 1, w = Tx[i], $\tau = t$
10:      /* Check if unassigned transceivers are available across all stars */
11:      elseif( $Tx[i,t]==0$ && $Rx[j,t]==0$ && $|\lambda_{full}[i,w]| \neq 0$)
12:          Grant = 1, $w = \lambda$, $\tau$ = star
13:      /* Check if transmitters are unassigned across all stars */
14:      elseif($Tx[i,t]==0$ && $Rx[j,t]\neq 0$ && $|\lambda_{full}[i,w]| \neq 0$)
15:          Grant = 1, w = Rx[j,t], $\tau = t$
16:      /* Check if receivers are unassigned across all stars */
17:      elseif ($Tx[i,t]\neq 0$ && $Rx[j,t]==0$ && $|\lambda_{full}[i,w]| \neq 0$)
18:          Grant = 1, w = Tx[i,t], $\tau = t$
19:      else Grant = 0;
20:      endif
21:      /*If request is granted, update slots and wavelength registers*/
22:      if(Grant == 1)
23:          if(size(i,j) > |L|-slot[w, $\tau$]) /* limited by number of slots available */
24:              slot[w, $\tau$] = |L|, size(i,j) = size(i,j) – (|L|-slot[w, $\tau$])
25:              Grant = 0 /* Not all the slots that were requested were granted */
26:          elseif(size(i,j) $\leq$ |L|-slot[w, $\tau$]) /* Slots are available */
27:              slot[w, $\tau$] = slot[w, $\tau$]+ size(i,j)
28:              Q<i,j> = 0 /* Request is fully granted */
29:          endif
30:      endif
31:      if (slot[w, $\tau$] == |L| || $m_w[w, \tau]$== |M|)
32:          $|\lambda_{full}[w, \tau]| = 0$ /*Update wavelength full register */
33:      endif
34:      endif
35:  end
```

The heuristics need at least N*R clock cycles to complete one pass through all requests. In practice, we found that 10*N*R is required to maximize throughput with these techniques. Hence, although it is possible to implement these heuristics in serial hardware, around 80,000 clock cycles would be required to compute a full epoch schedule with good matching performance.

C. Parallel Hardware Scheduler (A3)

The parallel hardware scheduler algorithm, optimized to meet the timing requirements defined in section II, is discussed in detail here. As shown in figure 1, the first phase

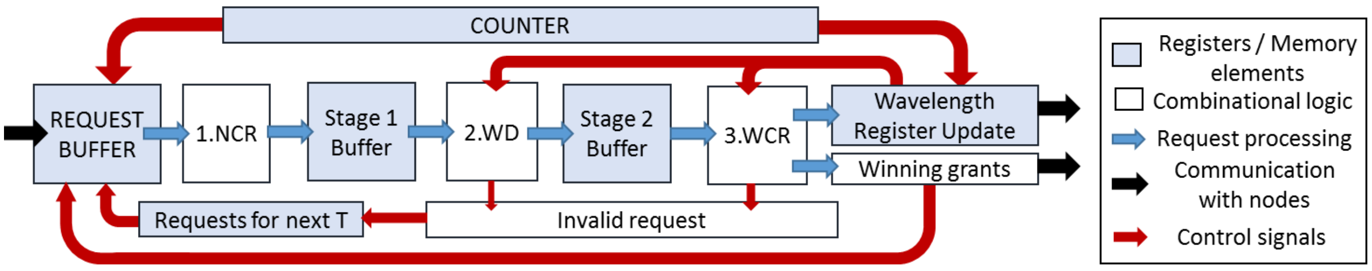


Figure 2: Scheduler algorithm with three stages: 1. Node Contention Resolution (NCR), 2. Wavelength Decision (WD) and 3. Wavelength Contention Resolution (WCR)

of the scheduler is the collection of requests, consisting of destination and number of TDM slots required, from N nodes. Once the request collection phase has completed, the scheduler performs multiple iterations on buffered requests. The scheduler must process requests from 1000 nodes, allocating timeslots (TDM) across 320 wavelength channels and 4000 transceivers (WDM) (assuming T=4 star couplers) within a time constraint of 1µs. To meet this time constraint, a high degree of parallelism is required, as serial looping algorithms will take thousands of iterations to compute a complete schedule. However, parallel schedulers introduce contention in the assignment of network resources in the same clock cycle. In general, situations in which there is contention for resources in the same clock cycle in a digital circuit are known as *hazards*. Our general approach to maximizing parallelism without hazards is to use round-robin arbiters to fairly select up to W requests with unique input port, output port and wavelength which can be assigned TDM timeslots in the same clock cycle without contention. The technique operates sequentially for each star coupler (I/T iterations), running the entire algorithm T times. After performing up to I iterations in 1µs, the scheduler sends the TDM grants and wavelength assignments to the nodes. There are three key sub-modules within the scheduler to achieve this, as shown in figure 2:

1) *NCR - Node Contention Resolution*

Considering one request from a node per iteration, the scheduler performs output node contention resolution in the NCR stage. In a parallel hardware algorithm, only requests with a unique source/destination pair can be granted in a single clock cycle. Hence, the NCR stage uses round robin (RR) arbiters to select one request per input/output node pair. The scheduler uses N parallel N-bit arbiters and generates up to N contention-free node pairs.

2) *WD - Wavelength Decision*

The wavelength decision block works on contention-free node pairs, from the Stage 1 buffer, and cross checks with the wavelength registers, to verify if the transmitter or receiver in question has already been assigned any wavelengths on previous iterations. If contradicting wavelengths have already been assigned, the request is invalidated for the current epoch. If the current wavelength assignments are consistent with fulfilling the request, the request moves to the WCR stage. If both the transmitter and the receiver have never been granted a wavelength before, then a random wavelength is chosen.

3) *WCR - Wavelength Contention Resolution*

The wavelength contention resolution block considers the wavelength requests made by the WD sub-module and resolves wavelength contention, allowing a maximum of W grants to be successful in any iteration. The WCR module uses W N-bit round robin arbiters to resolve contention. The successful requests are granted and assigned all the required TDM timeslots. Any new wavelength assignments are stored.

IV. Implementation of the Hardware Scheduler

In this section, we demonstrate that the proposed parallel hardware scheduler is viable for implementation on a CMOS ASIC. First, the scalability of round robin arbiters, one of the key circuit blocks, is evaluated. Following this, we describe the implementation techniques used to enhance the speed of the scheduler. Finally, we review the scalability, timing and area characteristics of the different parallel scheduler sub-modules on 45nm CMOS technology.

A. *Arbiter Scalability*

A key component of our parallel hardware scheduler is the round robin arbiter. The arbiters are used to fairly identify non-contending destination requests and non-contending wavelength requests for scheduling. As the arbiter is in the critical path for large port counts, to enable a scalable scheduler design overall, it is very important that the arbiters are scalable. The round robin arbiter, based on the design proposed in [15], was coded in SystemVerilog hardware description language with configurable size and look-ahead value. An N-port round robin arbiter consists of a circular programmable priority encoder (PPE) in which the priority is given to the last granted port plus one (the last granted port has the lowest priority). The critical path in the arbiter is through the carry chain in the PPE, rather like the carry chain in an adder circuit. As for adders, carry look-ahead can be employed to reduce the critical path. Unlike adders, the carry chain is circular, starting at the priority request cell and extending through N cells. The work in [15] shows how to implement the arbiter without using circular PPEs. The arbiter code was synthesized for various values of port-count and look-ahead using Synopsys Design Compiler and the 45nm CMOS NanGate standard cell library to extract timing and area characteristics. Figure 3 shows the best clock period achieved for arbiters with between 2 and 1024 ports for carry look-ahead chain of different lengths. The results show that a 1024-bit arbiter needs a minimum clock period of 4.7 ns, when using an optimum look-ahead of 16. The 1024-bit arbiter occupies an area of 0.031mm^2 on 45nm ASIC.

The critical path of the round robin arbiter scales sub-linearly with port count and hence appears promising for the implementation of large port-count schedulers.

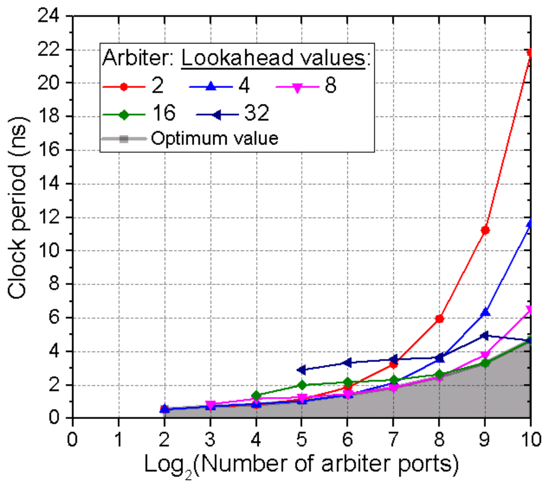


Figure 3: Scalability of an N-port arbiter on 45nm ASIC where N is the number of servers

B. Complexity Analysis and Synthesis Results

Each of the three scheduler sub-modules, discussed in section III C, have different complexity levels. The NCR block requires the implementation of N×N-bit arbiters; this has an area/component scaling of O(N^2), although the critical path length scales much more slowly as demonstrated in the previous section. The WD block is based on the implementation of N×(W:1) multiplexers with a complexity of O($N\log_2 W$). The third WCR module uses W×N-bit arbiters to resolve wavelength contention and hence, has an area/component scaling of O(NW). As shown in figure 2, each of the 3 sub-modules is placed in a separate pipeline stage to minimize the overall critical path.

The three sub-blocks, NCR, WD and WCR, were synthesized using Synopsys Design Compiler on 45nm CMOS technology using the NanGate standard cell library.

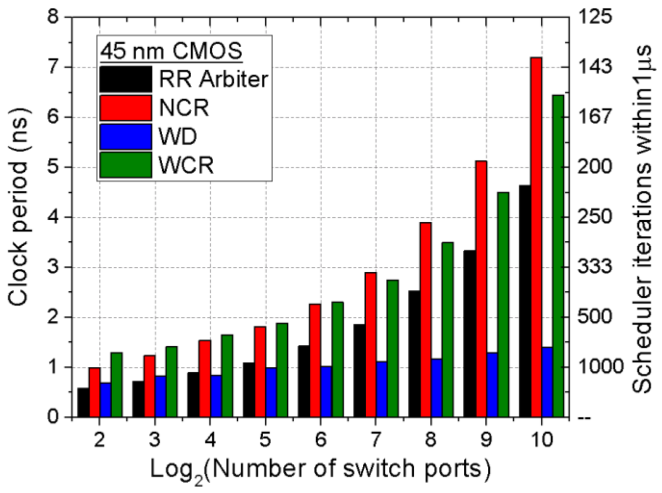


Figure 4: Scalability of N-bit NCR, WD and WCR sub-modules on 45nm ASIC, where N is the number of servers

Figure 4 shows the scalability of the sub-modules with respect to minimum clock period. As the arbiter is the dominant logic in determining the critical path of the NCR and the WCR stage, the optimized round robin arbiter results from figure 3 are also shown for comparison. Figure 4 shows that, in a 1000-port scheduler system, the NCR sub-module contains the overall critical path, meeting timing at a clock period of 7.2ns This corresponds to I=138 scheduler iterations which can be carried out in 1µs (shown in figure 4 by the right-hand y-axis). The total area consumed by the three scheduler sub-modules, using the 45nm standard cell library without considering other peripheral circuits such as SERDES, is 52.7mm^2. For comparison, a typical 64 x 10 Gb/s switch die consumes an area of 200-400mm^2 in 45nm CMOS technology [16]. These scalability results show the feasibility of scheduler implementation on digital hardware.

Given the large number of scheduler iterations possible, to avoid duplication of the scheduler logic, we propose to time share the scheduler circuit between stars. For example, for T=4, each star uses one quarter of the iterations available per epoch. The performance results in section V take into account this time sharing.

V. Performance Analysis

In this section, we compare the performance of the parallel hardware scheduler with ideal maximal matching and the software heuristics. The three algorithms, along with the traffic generators described in subsection A below, were coded in MATLAB. To increase confidence in our models and design, we simulated the SystemVerilog code for the hardware scheduler using the digital design simulator, Mentor Graphics Modelsim, importing the scheduler request inputs from MATLAB, and verified that the outputs were consistent. The switch parameters used for performance analysis are as follows. The number of ports in the switch, N = 1000. The maximum number of requests that can be made per node, per epoch, R = 8 initially. The average number of timeslots requested (per request) is 4. The number of wavelength channels, W = 80. The number of timeslots per epoch, L = 100. The number of stars or the number of transceivers per node, T = 4. 100% offered load (in Fig. 5) corresponds to making requests for the total capacity of the switch (32Tb/s).

A. Traffic Patterns

Three different traffic patterns were used to analyze the matching performance of the wavelength allocation schemes. The traffic patterns mentioned below generate requests for up to 30 epochs; the results are averaged and presented.

Uniform Random: Under uniform random traffic, each node requests a destination node with a probability of 1/N. Figure 5(a) shows the performance of the scheduling algorithms under uniform traffic for increasing network load. At lower loads, all three scheduling algorithms grant close to all offered load. As the offered load increases, all algorithms achieve less matching, as expected. At 100% offered load, the hardware scheduler has 10% lower matching than the maximal matching case. However, the parallel scheduler performs better by 8-10% compared to the serial scheduling heuristics. The parallel scheduler collects non-conflicting assignments to minimize the effect of hazards. In addition to this, the parallel and maximal matching algorithms aim to maximize resource allocation in each star, enhancing the throughput; the hazards are collected and processed in the next star.

Hotspot Traffic: In this case, a specific number of hotspot nodes request a percentage of the resource. This scenario simulates heavy application traffic patterns where few nodes request a large portion of the switch capacity. Figure 5(b) shows the performance of the scheduling algorithms in hotspot traffic patterns, when 5% of nodes are hotspots (50 nodes in the 1000-port system). An increased presence of hotspot traffic decreases the overall throughput, even for the maximal matching algorithm. This occurs because the number of hotspots is less than the number of wavelengths. For example, at 100% load, although there are 80 available wavelength channels in the network, only 50 destination nodes are requested repeatedly. The parallel scheduler performs worse by 12% compared to least loaded and least used allocation techniques and by 15% compared to maximal matching. Efficient matching performance in hotspot traffic requires careful allocation of available resources. Hence, least loaded, least used and maximal matching algorithms achieve a higher throughput compared to random wavelength selection in the parallel scheduler and the serial random scheme.

Clustered Traffic: In clustered traffic, the network is partitioned into several clusters and sets of nodes request only within their cluster. This simulates the scenario in which several nodes are connecting servers working on the same application. Figure 5(c) shows the performance of the scheduling algorithms in clustered traffic. There is an increase in the performance of all algorithms as the number of clusters increases, as sub-networks can be mapped onto the same wavelength(s). Achieving higher throughput with more clusters, the parallel scheduler algorithm achieves close to maximal matching when more than 50 clusters exist in the network.

B. Latency vs Load

To measure the latency using the three scheduling algorithms, we carried out simulations for the uniform random traffic pattern for up to 50 epochs, in which requests that were not granted in the current epoch are buffered and carried forward to the next epoch, adding to new requests. A buffer system was introduced to the scheduler algorithms to store the requests that were not granted in the current epoch. In consequent epochs, the buffer has priority to the requests from nodes; the requests are not invalidated completely until granted.

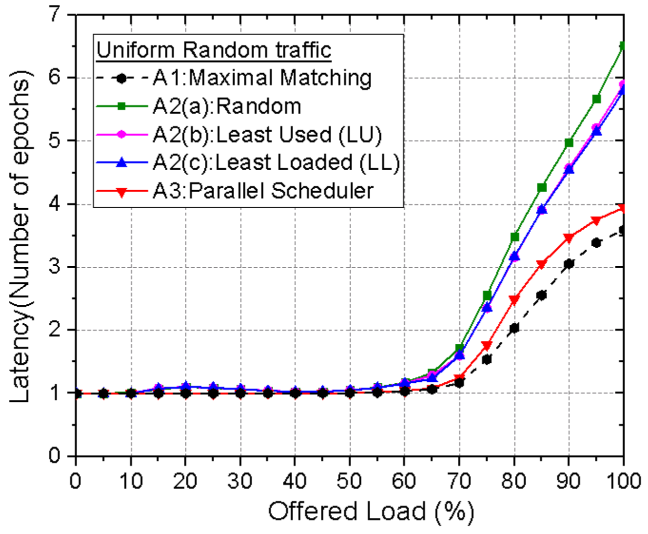


Figure 6: Latency of the scheduling algorithms

Figure 6 shows the performance of the scheduling algorithms in terms of packet latency for increasing network load. The minimum latency for any packet is one epoch. The load saturation point for the scheduling algorithms occurs between 60-70%, beyond which, the packet latency increases. The parallel scheduler and maximal matching scheme perform better than the software allocation schemes. The overall packet latency in the parallel scheduler is less than 4 epochs, which corresponds to 8.8μs (including 200ns tuning time for each epoch), for the given system parameters, even at 100% offered load.

C. Number of Requests per Node

The results obtained so far are presented for a scenario where the number of requests per node, R, is limited to 8. A practical scheduler implementation can only accept a fixed number of requests, constrained by the bandwidth of the control plane, scheduler memory and area consumption. However, to test the limits of the scheduling algorithms, the performance was evaluated for different values of R. In every epoch, the scheduling algorithms receive a maximum of N*R requests.

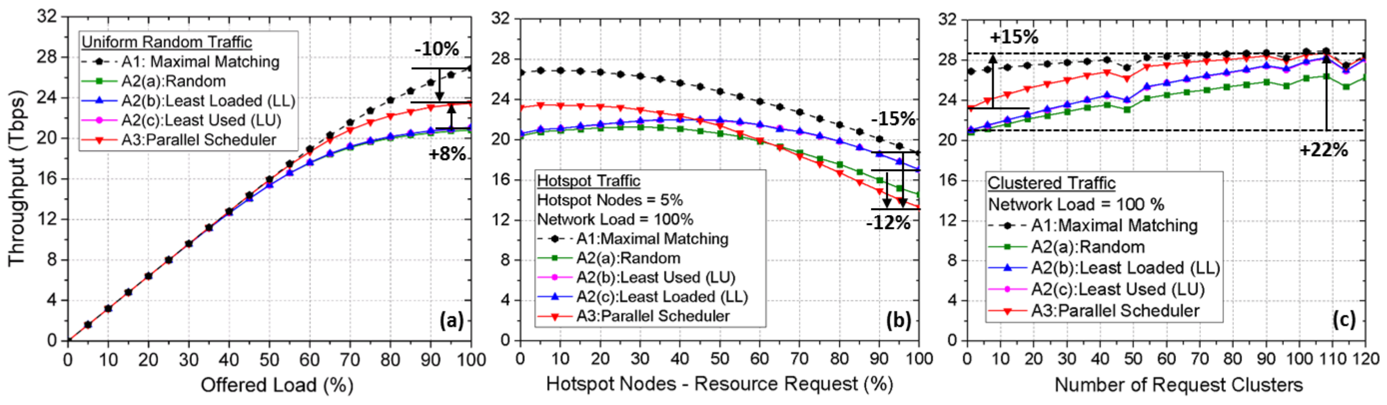


Figure 5: Performance analysis of scheduling algorithms under different traffic patterns

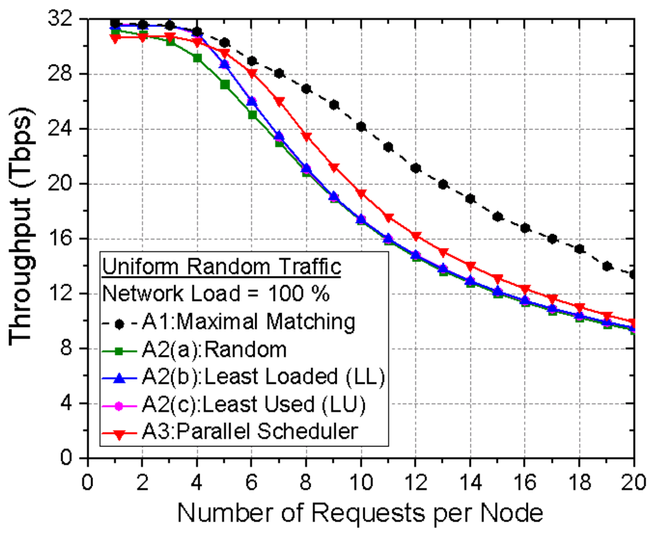


Figure 7: Performance of scheduling algorithms in requests per node

As shown in figure 7, as the number requests per node increases, there is a decrease in throughput for all scheduling algorithms. In these measurements, the request capacity for the switch resources was kept the same (32 Tb/s). Lower numbers of requests represent a high concentration of elephant flows, where each request is for large number of slots. The higher numbers of requests correspond to a high concentration of mice flows, where many small slots are requested. In this scenario, when R increases to high values the matching performance decreases due to the increasing number of hazards. The parallel scheduler algorithm performs worse by 10-15% compared to the ideal maximal case, when the number of requests is greater than 10. The parallel scheme also achieves a small performance gain compared to serial software scheduling schemes.

D. Summary

In summary, the parallel hardware scheduler achieves an equivalent matching performance to serial software wavelength allocation techniques on packet timescales. The load-latency results in section V.B show that the parallel scheduler performs better than conventional schemes by 2-3 epochs, making an efficient use of each star available. Finally, as expected, the scheduling algorithms are shown to achieve higher throughput for lower number of requests, showing the limitation of the optical switch plane.

VI. Conclusion

The scalability of data center optical switches is limited by the complexity of the control plane. The optical switch architecture demonstrated in [9] scales to over 1000 ports on the data plane but requires a scheduler performing dynamic wavelength assignment that is also scalable to 1000 ports and also achieves high throughput consistent with packet timescales. In this work, we presented a parallel hardware scheduler to meet these requirements. The implementation feasibility of the parallel scheduler in 45nm CMOS technology was demonstrated. The scheduler design was found to consume 52.7mm^2 ASIC area, relatively smaller than a conventional 64-port electronic network switch ASIC. The hardware scheduler was shown to perform up to 138 iterations within 1μs achieving matching performance close to that of maximal matching for various traffic patterns. The hardware scheduler also outperforms conventional wavelength assignment heuristics which would take on the order of milliseconds to execute in most cases.

VII. References

[1] W. Paper, “Cisco Global Cloud Index : Forecast and Methodology ,” pp. 2015–2020, 2016.

[2] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, “Reconfigurable network systems and software-defined networking,” *Proc. IEEE*, vol. 103, no. 7, pp. 1102–1124, 2015.

[3] T. P. Morgan, “Broadcom Strikes 100G Ethernet Harder With Tomahawk-II,” 2016. [Online]. Available: https://www.nextplatform.com/2016/10/31/broadcom-s. [Accessed: 08-Feb-2017].

[4] S. Di Lucente, N. Calabretta, J. a. C. Resing, and H. J. S. Dorren, “Scaling Low-Latency Optical Packet Switches to a Thousand Ports,” *J. Opt. Commun. Netw.*, vol. 4, no. 9, p. A17, Jul. 2012.

[5] K. Xi, Y. Kao, M. Yang, and H. J. Chao, “Petabit Optical Switch for Data Center Networks,” New York City, NY, 2010.

[6] J. Gripp, J. E. Simsarian, J. D. LeGrange, P. Bernasconi, and D. T. Neilson, “Photonic terabit routers: The IRIS project,” *Opt. Fiber Commun. (OFC), collocated Natl. Fiber Opt. Eng. Conf. 2010 Conf.*, pp. 1–3, 2010.

[7] C. Kachris and I. Tomkos, “A Survey on Optical Interconnects for Data Centers - IEEE Communications Surveys & Tutorials.pdf,” vol. 14, no. 4, pp. 1021–1036, 2012.

[8] A. Shacham and K. Bergman, “An Experimental Validation of a Wavelength-Striped, Packet Switched, Optical Interconnection Network,” *J. Light. Technol.*, vol. 27, no. 7, pp. 841–850, 2009.

[9] D. Alistarh, H. Ballani, P. Costa, A. Funnell, J. Benjamin, P. Watts, and B. Thomsen, “A High-Radix , Low-Latency Optical Switch for Data Centers,” *Proc. ACM SIGCOMM (Poster/demo Sess.*, pp. 367–368, 2015.

[10] B. Rahimzadeh Rofoee, G. Zervas, Y. Yan, and D. Simeonidou, “Griffin: Programmable Optical DataCenter with SDN Enabled Function Planning and Virtualisation,” *J. Light. Technol.*, vol. 33, no. 24, pp. 5164–5177, 2015.

[11] A. Funnell, J. Benjamin, H. Ballani, P. Costa, P. Watts, and B. C. Thomsen, “High Port Count Hybrid Wavelength Switched TDMA (WS-TDMA) Optical Switch for Data Centers,” *Opt. Fiber Commun. Conf. 2016*, pp. 2–4, 2016.

[12] S. Das, “A Tight Lower Bound for the Weights of Maximum Weight Matching in Bipartite Graphs,” pp. 1–9, 2016.

[13] M. Demange and T. Ekim, “Minimum maximal matching is np-hard in regular bipartite graphs,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4978 LNCS, pp. 364–374, 2008.

[14] H. U. I. Zang, “A Review of Routing and Wavelength Assignment Approaches for Wavelength- Routed Optical WDM Networks,” *Opt. Networks Mag.*, vol. 1, no. January, pp. 47–60, 2000.

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

[16] N. Farrington, E. Rubow, and A. Vahdat, “Data Center Switch Architecture in the Age of Merchant Silicon,” pp. 101–110, 2009.