Subchannel Scheduling for Shared Optical On-chip Buses

DOI:
10.1109/HOTI.2017.18

Document Version
Accepted author manuscript

Subchannel Scheduling for Shared Optical On-chip Buses

Sebastian Werner, Javier Navaridas and Mikel Luján
School of Computer Science, The University of Manchester, Manchester, UK
Email: {sebastian.werner, javier.navaridas, mikel.lujan}@manchester.ac.uk

***Abstract*—Maximizing bandwidth utilization of optical on-chip interconnects is essential to compensate for static power overheads in optical networks-on-chip. Shared optical buses were shown to be a power-efficient, modular design solution with tremendous power saving potential by allowing optical bandwidth to be shared by all connected nodes. Previous proposals resolve bus contention by scheduling senders sequentially on the entire optical bandwidth; however, logically splitting a bus into subchannels to allow both sequential and parallel data transmission has been shown to be highly efficient in electrical interconnects and could also be applied to shared optical buses.**
In this paper, we propose an efficient subchannel scheduling algorithm that aims to minimize the number of bus utilization cycles by assigning sender-receiver pairs both to subchannels and time slots. We present both a distributed and a centralized bus arbitration scheme and show that both can be implemented with low overheads. Our results show that subchannel scheduling can more than double throughput on shared optical buses compared to sequential scheduling without any power overheads in most cases. Arbitration latency overheads compared to state-of-the-art sequential schemes are moderate-to-low for significant bus bandwidths and only noticeable for low injection rates.

I. Introduction

Optical networks-on-chip (ONoCs) are widely considered a promising candidate for future power-efficient on-chip communication. However, static power required at the laser source and for microring (MR) heating considerably degrades their overall efficiency and poses a major obstacle that has been under ongoing investigation both on the technology and architectural level in recent years. Although optical interconnects can provide high bandwidth density thanks to dense wavelength-division multiplexing (DWDM), both laser and heating power significantly increase along with link bandwidth, i.e. the number of wavelengths. This represents a critical issue in the on-chip domain since many multi-threaded applications exhibit compute-intensive execution phases in which they underutilize the NoC, and at the same time require high bandwidth in communication-intensive phases [1]. Static power would thus be wasted in low-utilization phases, but at the same time cannot be avoided due to the bandwidth demands in high-utilization phases. Architectures utilizing the available optical bandwidth efficiently are therefore of high interest.
Shared optical buses were shown to be highly efficient in terms of bandwidth utilization [2]. Like in traditional electrical buses, multiple nodes are connected to a bus, perform bus arbitration prior to data transmission, and transmit data through time-division multiplexing (TDM); however, optical buses do not have the same limitations as electrical buses: distance does not play a major role in optical data transmission due to almost distance-independent dynamic energy consumption and signal propagation delay of light in silicon. Besides, bandwidth can be scaled much more efficiently through DWDM within the same waveguide (rather than adding wires on electrical buses). State-of-the-art optical bus arbitration proposals schedule nodes that simultaneously request the bus sequentially on the entire optical bandwidth [2], and senders/receivers tune in their modulators/filters in their assigned timeslots. We argue that large throughput improvements could be achieved by allowing multiple sender-receiver pairs to utilize the bus both sequentially *and* in parallel on different subchannels of the available wavelengths on the bus. Leveraging subchannels on the same physical channel is a concept well-known in electrical interconnects [3][4] and could also be adopted in shared optical buses since MRs can be tuned/detuned to respond to different wavelengths individually (or in groups), allowing for logically dividing the optical bus into subsets of wavelengths. We believe that this scheduling approach could tremendously improve throughput and power efficiency of shared buses if the arbitration mechanism can be implemented with low overheads in terms of latency and power. This paper tackles the task of finding an efficient subchannel scheduling algorithm and low-overhead arbitration schemes, and makes the following **contributions**:

- An easy-to-implement subchannel scheduling algorithm that achieves high bus utilization and is adaptable to any bus bandwidth, bus size, number of subchannels, and flow control mechanism.
- A centralized and a distributed arbitration scheme that implement subchannel scheduling with negligible power overheads. Arbitration latency overheads are moderate-to-low and only noticeable at low injection rates.
- Compared to the state-of-the-art sequentially-scheduled bus LumiNOC [2], our proposal doubles throughput without incurring any power overheads for significant bus sizes and bandwidth.

II. The Shared Optical Bus

On-chip buses enable all-to-all communication between all connected nodes. In the optical domain, these have conventionally been implemented by assigning either one Single-Writer-Multiple-Reader (SWMR) or Multiple-Writer-Single-Reader (MWSR) bus to each node to form a crossbar [5].

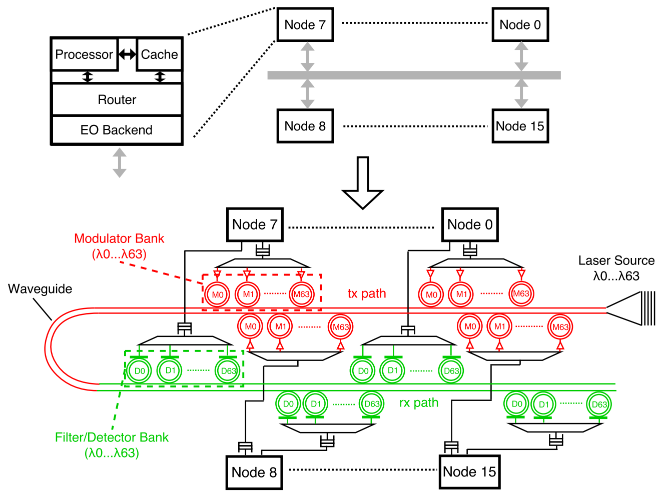


Fig. 1: Shared Optical Bus

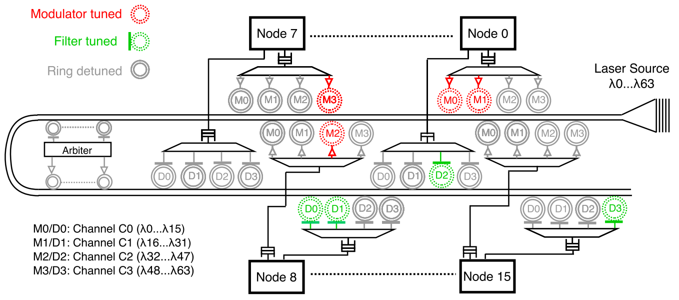

Modulator tuned
Filter tuned
Ring detuned
Node 7
Node 0
Node 8
Node 15
Arbiter
Laser Source λ0...λ63
M0/D0: Channel C0 (λ0...λ15)
M1/D1: Channel C1 (λ16...λ31)
M2/D2: Channel C2 (λ32...λ47)
M3/D3: Channel C3 (λ48...λ63)

Fig. 2: Utilization example of a shared optical bus during data transmission phase with subchannels

The major drawback of these designs is that both wavelengths (λ) and the number of links scale linearly with the number of nodes, leading to limited scalability and large static power.

A. *Motivation*

The shared optical bus (see Fig. 1) has been proposed to tackle these problems by connecting all N nodes to *one* waveguide on which they share the available optical bandwidth ($\lambda_0..\lambda_{63}$) [2]. This significantly reduces the total number of λ and in turn laser power and scalability. Besides, it reduces the number of waveguides from N to 1. Enabled by a U-shaped waveguide, each node modulates on the transmit side (tx path, in red) and receives on the receive side (rx path, in green). In contrast to a crossbar, this approach requires TDM to avoid data corruption when two nodes transmit simultaneously. Therefore, shared optical buses work in two phases: An arbitration phase in which nodes request the bus and are granted access by an arbitration mechanism; and a data transmission phase in which nodes transmit data in their assigned timeslot. If multiple senders request the bus simultaneously, bus arbitration and scheduling is required.
Sharing wavelengths on the same waveguide is enabled by the capability of controlling the resonance wavelength of MRs– the nanophotonic building block for modulators and filters– through MR tuning: dynamically decide to either filter a certain wavelength or let it pass. For instance, if node 0 owns the bus and wants to send to node 7, node 7 would tune in its filters while all other nodes keep theirs detuned.

III. Subchannel Scheduling

State-of-the-art proposals of shared optical buses resolve bus contention by scheduling all requesting nodes sequentially on the entire optical bandwidth [2]; however, MRs typically have integrated heaters [6] and can be tuned/detuned either individually or in groups, which would allow to schedule requesting nodes both sequentially and in parallel on different λ-subsets – *subchannels*. In this section, we introduce our subchannel scheduling approach, discuss the latency of optical data transmission, and analytically show the superiority of our approach compared to sequential scheduling.

A. *Bus Splitting into Subchannels*

Subchannels are formed by splitting the optical bandwidth available on the bus *logically* into non-overlapping subsets. This allows multiple sender-receiver pairs to communicate simultaneously on the same bus by utilizing different subchannels. For instance, in Fig. 2, node 0 sends to node 8 on subchannel C0 and C1, while node 7 sends to node 15 on subchannel C3, and node 8 to node 0 on C2. This is enabled by each node tuning in sets of modulators (M1-M3, in red) and filters (D0-D3, in green) according to their assigned subchannel(s), while detuning all other MRs (gray). In the example, the optical bandwidth of 64λ is divided into four subchannels with 16λ each, i.e each set of modulators and filters consists of 16 MRs each. To identify the ideal number of subchannels and bus widths, we will now study the components that add to the total latency on optical links.

B. *Minimizing Bus Utilization Cycles*

Optical data transmission includes 1) electrical-to-optical (E/O) data conversion through modulation, 2) signal propagation delay on the waveguide, and 3) optical-to-electrical (O/E) data conversion through detection. E/O depends on the core frequency, modulation rate, and the number of λs available for modulation. Assuming a modulation rate of 10 Gb/s and 5 GHz core frequency [5], 2 bits can be modulated on one λ per clock cycle, or, for instance, 64-bit could be modulated in two clock cycles by 16λs. Signal propagation of light in a waveguide is estimated to take 10.45 ps/mm [7], which means, at a 5 GHz clock (i.e. 200 ps cycle duration), the propagation delay on optical links <19 mm is one processor clock cycle, which is a wide range for common tile widths of 1-2 mm [8]. Delay in the detector and O/E backend circuitry often adds another clock cycle [9].
Tuning speeds of MR resonators–i.e., the time it takes to shift and stabilize their resonance wavelength–have been subject to extensive research [10][11][12]. Previous studies assume tuning to be executed within one core clock cycle at 5 GHz [13]. Tuning times lower than 500 ps have been reported [11], which would take up to 3 clock cycles at 5 GHz. As this is a worst case upper limit, and faster tuning times are expected, we

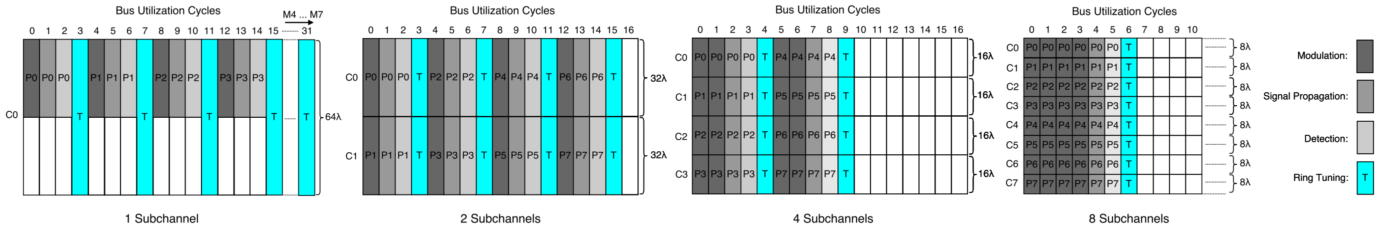


Fig. 3: Data transmission phase on a bus for 64-bit data packets with varying numbers of subchannels.

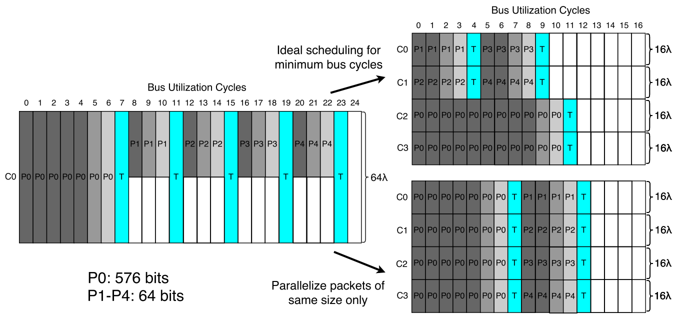

Bus Utilization Cycles
Ideal scheduling for minimum bus cycles
Parallelize packets of same size only
P0: 576 bits
P1-P4: 64 bits

Fig. 4: Scheduling Multiple Packet Sizes

assume tuning delays of one clock cycle. Longer tuning delays, however, would further increase the efficiency of subchannel scheduling compared to sequential scheduling as in total fewer consecutive tuning cycles are required compared to sequential scheduling only (see Fig. 3).

These latencies are illustrated in Fig. 3, which depicts the way the bus timeslots are utilized for transmitting several 64-bit packets[1]. The leftmost example demonstrates sequential data transmission over *one* channel comprising the entire bandwidth. Each packet (P0, P1, etc.) occupies the bus for 4 cycles (including MR tuning delay between the packets). Note that, for a 64-bit packet, only 32λ are needed for modulating it in one clock cycle, effectively wasting half of the bandwidth in this case–one of the weaknesses of strict sequential scheduling. As we increase the number of subchannels to 2, 4, and 8 we notice a positive effect on the total utilization cycles. Halving the bandwidth of each subchannel from e.g. 32λ (2 subchannels) to 16λ (4 subchannels) leads to twice the modulation latency, so the overall modulation time in the parallel and sequential case is the same; however, propagation, detection, and MR tuning overheads are parallelized, which leads to large latency savings overall. For instance, in the 8 subchannel case, propagation/detection/tuning delay of each packet is only impacting the total latency *once*, while e.g. in the 1 subchannel case, this latency is added up for every single packet scheduled on the bus. These overheads are independent from – and thus applicable to – any packet size. The ideal case, from an analytic perspective, is to provide one subchannel for each node attached to the bus (i.e. maximum number of simultaneous requests), with a total bus bandwidth that is divisible by N (to prevent uneven subchannel widths).

[1]Packet sizes in chip multiprocessors typically vary between 64-bit for simple requests and coherence traffic, and 576-bit for cache line transfers. We chose 64-bit in this example for simplicity/illustration purposes.

C. Subchannel Scheduling Algorithm

The previous subsection revealed the benefits of scheduling packets both in time slots and on subchannels. We will now discuss our light-weight subchannel scheduling algorithm that achieves near-minimal bus utilization with a simple, low-overhead implementation, and flexible enough to accommodate to any bus widths, number of subchannels, number of nodes, and flow control mechanism.

Since scheduling computation is performed in the arbitration phase, it should ideally take only one cycle to determine an efficient scheduling of the incoming requests. While computing the minimal latency for nodes requesting the bus for the same packet size is simple and straight-forward, this is not the case if the bus is requested for multiple packet sizes, e.g. 64 bit and 576 bit, which is an NP-hard problem that could potentially take considerably longer to compute than one clock cycle.

Fig. 4 illustrates this: Assume the bus is requested for one 576-bit packet (P0) and four 64-bit packets (P1-P4) and that four subchannels are available (on the right). On the left, sequential scheduling takes 24 cycles in total. The top right figure depicts the ideal scheduling of these packets which schedules packets of different sizes in parallel and minimizes the latency down to 12 cycles. This requires to determine all possible combinations of these packets on all possible time slots and subchannels, leading to a computation complexity that has factorial growth with the number of requests.

During our study, we identified that grouping requests according to the packet sizes they are requesting the bus for and parallelizing only packets of the same size tremendously simplifies the scheduling algorithm and only leads to small bus cycle overheads that are neutralized by the latency saved during scheduling computation in the arbitration process. The bottom right figure in Fig. 4 illustrates this scheduling, which, in this example, sends the 576-bit packet on the entire bandwidth and parallelizes the 64-bit packets subsequently, resulting in just one additional cycle. While this is a simple example for illustration purposes, this trend was observed to hold true for higher quantities of requests and combinations.

Alg. 1 shows the algorithmic definition of our allocation circuitry. This extends to multiple packet sizes by having a separate queue for each and applying the allocation separately. At this point we make no assumption on which packet sizes should go first, and leave this for future analysis as our

ALGORITHM 1: Subchannel and Slot Allocation

```
Queue sorted_reqs = sortReqsBasedOnCredits(requests_in);
while (!sorted_reqs.empty()) do
    if (num_reqs >= num_subchannels) then
        num_current_reqs = num_subchannels;
    else
        num_current_reqs = sorted_reqs.size();
    #SC = num_subchannels / num_current_reqs;
    for (int i = 0; i < num_current_reqs; i++) do
        assign(sorted_reqs.pop(), C(i*#SC, #SC*(i+1)-1),
          slot_start_cycle);
    end
    slot_start_cycle += current_slot_duration;
end
```

focus is on maximizing bus utilization. Our greedy algorithm attempts to schedule as many packets in parallel as possible. If there are more requests than subchannels, each subchannel is assigned to a different packet, and the remaining packets will be scheduled in the next time slot. The pointer to the starting point of the time slot in Alg. 1 is the ‘slot_start_cycle’ variable, which is determined by the time occupied by the current (and all previous) time slots (‘current_slot_duration’) based on the packet size and number of λ, subchannels, and requests. For instance, in the ‘2 Subchannels’ example in Fig. 3, the first slot would start at cycle 0, the second at 4, the third at 8, etc. Before time slot and subchannel allocation, the incoming requests are stored in a queue (‘sorted_reqs’) based on their priorities/credits. This ensures that the flow control mechanism is obeyed by ensuring priorities are maintained. If there are less requests than subchannels, i.e. not all subchannels can be filled with requests, the optical bandwidth must be assigned evenly to minimize bus latency. The number of subchannels assigned to each requester (‘#SC’) is thus the total number of subchannels divided by the number of requests–e.g., if only one request remains it will use all subchannels, if there are $< \#subchannels/2$ requests each will use two subchannels, $< \#subchannels/4$ requests each will use four subchannels, etc. Each allocation is executed by the function *assign(request, subchannel-range, starting slot)*.

IV. Bus Arbitration Mechanisms

The arbitration phase is the default state of the bus and is entered by each node once data transmission is over and the bus free again. Arbitration can be centralized or distributed, with both approaches entailing different opportunities and trade-offs. In both cases, the exchange of control messages manages bus access. After the arbitration phase, each node must know the time slot and subchannel(s) on which it 1) is allowed to send its data (in case it contended for sending on the bus), 2) has to tune in its MR filters (in case it is a receiver) 3) has to keep its filters detuned (otherwise).

Control messages should be small while carrying enough information to enable correct scheduling of time slots and subchannels. Section III showed that sequential scheduling is inefficient; however, it simplifies arbitration as allocation is unidimensional (time slots only, not subchannels). In this section, we introduce both a centralized and distributed arbitration scheme for subchannel scheduling that exhibit low overheads compared to the sequential approach.

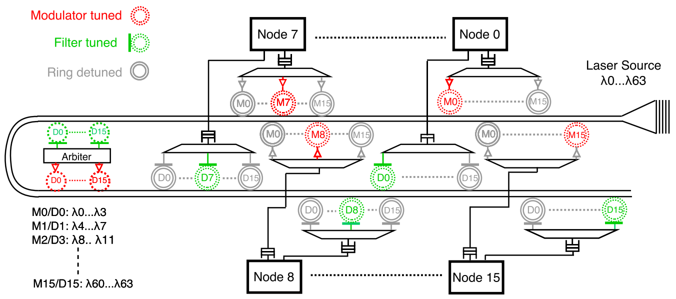


Fig. 5: Optical bus during arbitration phase

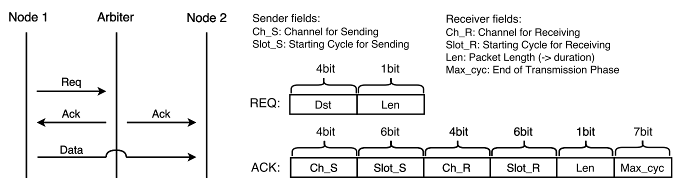
A. Centralized Arbitration

In centralized arbitration, an arbiter computes the scheduling and notifies all nodes about when and how they can access the bus, implemented by exchanging request (REQ) and acknowledgment (ACK) packets between the nodes and the arbiter. We propose to connect the arbiter to the bus as shown in Fig. 2, and perform arbitration and data transmission on the same optical bus, which allows to reuse optical resources for arbitration as opposed to having a separate control network.

At the beginning of the arbitration phase, nodes request the bus by simultaneously sending a REQ to the arbiter on a unique subset of $(bus\_width/\#Nodes)$- wavelengths. For instance, in Fig. 5, a bus of 64λ bandwidth and 16 nodes would provide each node with 4λ for modulating its REQ ($\lambda_0..\lambda_3$ for node 0, $\lambda_4..\lambda_7$ for node 1, etc.). The arbiter has filters to receive REQs from each node tuned to the according λ-subsets. *Requests* sent from the nodes to the arbiter contain fields indicating the packet’s destination ID (Dst) and length (Len). The source ID is implicitly known by the arbiter as each node sends on a unique set of λs. For N nodes and S packet sizes, Dst is $log_2(N)$ and Len $log_2(S)$ bits long. Once the arbiter received all REQs, it knows all senders, receivers, and packet sizes - all the information needed to compute subchannel slot allocation. This allows for a very compact REQ size that can be modulated quickly with little optical bandwidth. *Acknowledgments* are sent from the arbiter to each node on their according λ-subsets upon scheduling computation in order to notify senders about their subchannel time slots and receivers about when to tune/detune their filters. In addition, *all* nodes are informed about when the next arbitration phase begins. An ACK contains different fields based on the node’s role in the transmission phase:

Senders ACKs contain 1) a subchannel bitmap with the assigned subchannel(s) set to ‘1’ and 2) the time slot when they have to start sending.

Receivers ACKs contain 1) a similar subchannel bitmap, 2) the time slot to tune in, and 3) the packet length used by the receivers to compute the duration of the time slot, i.e. when they have to detune.



Node 1
Arbiter
Node 2
Req
Ack
Ack
Data
Sender fields:
Ch_S: Channel for Sending
Slot_S: Starting Cycle for Sending
Receiver fields:
Ch_R: Channel for Receiving
Slot_R: Starting Cycle for Receiving
Len: Packet Length (-> duration)
Max_cyc: End of Transmission Phase
REQ:
4bit
1bit
Dst
Len
ACK:
4bit
6bit
4bit
6bit
1bit
7bit
Ch_S
Slot_S
Ch_R
Slot_R
Len
Max_cyc

Fig. 6: Control message exchange in centralized arbitration and all possible REQ/ACK fields

Nodes sending and receiving within the same arbitration round will receive all info appended, e.g. if a node is receiving i packets, i receiver fields are appended. The sender fields will always be sent first as each node knows if it is a sender. Receiver fields are appended to the sender fields as needed.

Fig. 6 shows example REQ/ACK packets for a 64λ bus with 16 nodes – e.g, the ACK encodes a node sending and receiving one packet. While REQs have constant size, ACK size depends on the bus utilization rate (more requests lead to more fields being sent) and on the traffic pattern (multiple receptions in the same node require many receiver fields). With the information contained in the ACKs, every node knows when and on which subchannel(s) it can send, and when and to which subchannel(s) it has to tune in its filters. During data transmission phases, the arbiter detunes its MRs to avoid filtering the optical signals, and only tunes them in again for the next arbitration phase.

The 'max_cyc' field is appended to indicate when the next arbitration phase starts, and will be sent to every node. For instance, for 16 nodes, this could be up to 84 cycles if each node requests the bus for a 576 packet, in which case this field would be 7 bits. In case none of the nodes sends a REQ in an arbitration phase, the arbiter sends an ACK to each node with the 'max_cyc' field set to '0', indicating that the next arbitration phase can start immediately. REQs and ACKs thus increase along with the number of nodes, and modulation delay depends on the number of λs per node.

A separate arbitration unit naturally imposes certain hardware overheads. An example arbiter design is depicted in Fig. 7. Keeping control packets small is therefore not only key to low-latency arbitration, but also reduces buffer area in the arbiter. Assuming REQ sizes of R bits and N nodes, this would require (N×R)-bit buffer space, which is negligible for relevant bus sizes (e.g. 8-16 nodes). The scheduling computation unit outputs the according ACK packets for each node and must therefore provide enough output buffers. Receiving and sending data on the optical bus also requires corresponding MRs and E/O and O/E backend circuitry.

B. Distributed Arbitration

Distributed arbitration does not require a separate arbiter, leading to savings in area and power. Not having a centralized unit, however, makes implementing more sophisticated arbitration schemes more challenging as more information has to be encoded in the control messages, potentially increasing arbitration latency and energy. Previous studies propose an efficient distributed mechanism for bus scheduling [2]; however, they are merely capable of supporting scheduling of nodes in time slots, and not subchannels. Our mechanism provides an efficient solution for both.

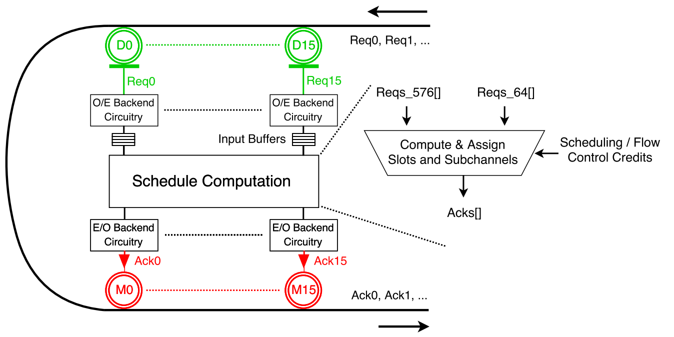


Fig. 7: Example arbiter design. Slots and subchannels are computed based on the incoming requests and assigned based on scheduling/flow control credits

Like in our centralized approach, each node is assigned to its unique set of λs in the arbitration phase. Each node *receives* control packets from other nodes on its own λ-set and modulates its control packets *on each of the other nodes' λ-sets*. Control packets are based on 1-hot encoded bitmaps where each bit represents one node. To provide each node with the necessary information to perform scheduling, two arbitration packets are sent subsequently in **two phases**:

1) Ctrl_1: [Src_Bitmap | Length_Bitmap]: Each node wishing to send broadcasts Ctrl_1 on the bus with its bit set to '1' in the Src_Bitmap field. At the same position in the Length_Bitmap field, it sets its bit to '1' to indicate a 576-bit packet, and leaves it to '0' for a 64-bit packet. All nodes must send Ctrl_1 simultaneously, so that correct bitmaps are created that carry the scheduling information.

2) Ctrl_2: [Src_Bitmap]: Every node wishing to send modulates a packet containing the same Src_Bitmap fields once again right after it sent Ctrl_1, but this time it will only modulate it on the λ-set assigned to its receiver, rather than broadcasting it. This allows receivers to identify their senders. As each receiver already knows the scheduling of the sending nodes from phase 1, it can look at this scheduling to see when it has to tune/detune its filters.

In phase 1, each node receives a bitmap that contains all sending nodes ('Src_Bitmap'), and another bitmap to indicate their packet sizes ('Length_Bitmap'). This suffices to compute our scheduling algorithm at each node, allowing each node to know the starting slot and subchannels for sending. In addition to that, information is required at each receiving node to know when MR filters have to be tuned/detuned. This information is provided in phase 2. After phase 1, each node knows the starting slot of each sender, the subchannel(s) it will send on, and the duration from the packet sizes. At this point, however, the receivers are not known. This information is obtained in phase 2, where each node will receive another 'Src_Bitmap'. If none of the bits in this bitmap is set to '1' at a receiver, this

means it will not receive any packets during this transmission phase and will keep its filters detuned. If one or more bits are set to '1', each receiver knows its senders as each bit is assigned to one node on the bus. Receivers can now look up in the scheduling results the senders' allocation of time slot and subchannel(s).
This arbitration should save power as no centralized arbiter with additional MRs and E/O and O/E circuitry is required, thereby reducing optical static power requirements. NoCs configured with more than two packet sizes (uncommon) would double the 'Length_Bitmap' in phase 1, potentially leading to considerable arbitration latencies. In this case, our centralized approach would offer higher efficiency and flexibility.

V. Evaluation

A. Methodology

In order to evaluate the efficiency of subchannel scheduling compared to sequential scheduling, we compare both our distributed and centralized arbitration mechanism to LumiNOC [2] in terms of latency, throughput, and power consumption. LumiNOC represents the state-of-the-art shared optical bus proposal and outperforms a large number of recently proposed ONoCs as well as electrical baselines [2]. It uses a distributed arbitration approach in which each requesting node broadcasts a bitmap with its respective source address bit set to '1' in case it would like access to the bus, along with a destination and packet length field to notify receivers to tune in. Leveraging a bus layout like in Fig. 2, each requester receives its own arbitration packet and detects, based on the bitmap, whether there are other nodes requesting. If so, each requester will be scheduled sequentially on the entire bus bandwidth based on a priority-based algorithm. LumiNOC reduces arbitration latency for cases with only one requester through speculation: nodes speculatively start sending data packets right after transmitting the control packet and only abort data transmission if multiple requesters are detected in the bitmap. A comparison to LumiNOC allows to identify the both overheads of more complex arbitration schemes and the benefits of subchannel scheduling.
We use DSENT [14] with a 22 nm technology library for energy and power estimations. Laser was extracted based on the loss parameters listed in Table I [15] and an off-chip laser with 25% wall-plug efficiency [16]. We assume 20 μW/MR heating power [9], 1 mm tile widths/lengths, 5 GHz core, router, and link clock frequency, and 10 Gb/s modulators/detectors. For performance simulations, we used the cycle-accurate simulator HNOCS [17], uniform random traffic, 256-bit packets, and 10000 packets per node injected with an exponentially-distributed inter-packet gap.

B. Latency and Throughput

Fig. 8 and Fig. 9 illustrate the average packet latency for buses utilizing our centralized (Centr) and distributed (Distr) approach compared to LumiNOC for varying bus bandwidth (i.e. number of wavelengths (λ)). The number of utilized subchannels is indicated by '#SCh'. We assume 10.45 ps/mm [7]

TABLE I: Optical Loss Parameters

Parameter	Value	Parameter	Value
Splitter loss	0.2 dB	Ring: Through	0.01 dB
Coupler	1 dB	Waveguide prop.	0.3 dB/mm
Ring: Drop	0.5 dB	Photodetector loss	0.1 dB

optical signal propagation delay, one cycle for OE, and one cycle for control packet processing.
Subchannel scheduling becomes increasingly superior with increasing number of λs and number of subchannels. LumiNOC shows smaller packet latency for low loads due to its speculation mechanism and less arbitration complexity. The differences to subchannel scheduling shrink as the number of nodes and λs decreases. As the injection rates increase, subchannel scheduling becomes increasingly superior, outweighing the arbitration overheads. Particularly as the number of nodes and wavelengths increases, subchannel scheduling improves throughput significantly (>1.6× for 64λ and >2× for 128λ for both bus sizes).

C. Power Consumption

Recent studies showed that there is a practical limit of $< 64\lambda$ within one waveguide [18] and that laser power increases tremendously along with the number of λs on a shared bus due to excessive ring-through losses [19]. We thus scale bandwidth on the buses by implementing 32λ buses and physically adding buses to increase bandwidth (logically, all nodes would still see the bus as *one* bus). For instance, a 128λ bus consists of four parallel 32λ buses. Leakage power includes the static electrical power for buffering arbitration packets and the E/O and O/E backends at the nodes and in the arbiter (in the centralized approach). Each node requires buffers to store one REQ and one ACK packet, and the centralized arbiter buffers for N REQs and N ACKs (for No nodes). We apportion a pessimistic 32-bit buffer for each control packet. Dynamic power was captured at the saturation point of LumiNOC (see Fig. 8/9)
Fig. 10 and Fig. 11 show the power breakdowns of 8-node and 16-node buses for the arbitration mechanisms under investigation for different bus widths (i.e. number of λs). As commonly known in ONoCs, static power consumed at the laser source and for MR heating is the major contributor to the total power, and our centralized arbitration approach exhibits slight overheads in terms of both these two metrics due to the additional number of MRs at the arbiter and the ring-through losses incurred by them. Moreover, the circuitry in the centralized arbiter causes higher leakage power. However, communication with the centralized arbiter allows for smaller arbitration packets and does not require broadcasting like in LumiNOC or our SCh_Distr. This translates to less arbitration energy, and in turn savings in dynamic power that cancels out the static power overheads of the centralized arbiter. In addition, dynamic power scales significantly better for larger bus sizes (see Fig. 11). This is because arbitration packets increase with $log2(\#nodes)$ in SCh_Centr, while both

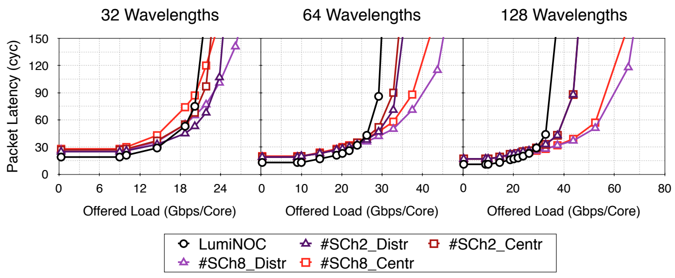


Fig. 8: Bus Latency 8 Nodes

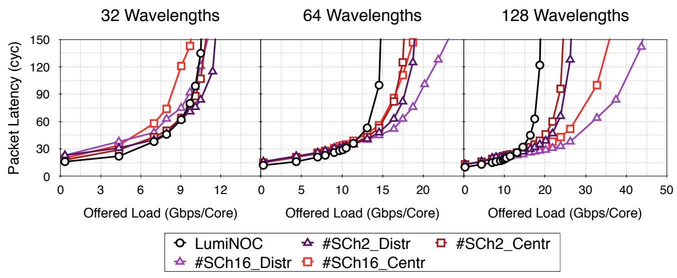


Fig. 9: Bus Latency 16 Nodes

LumiNOC and SCh_Distr require broadcasting bitmaps of the source addresses and thus scale with $\#nodes$.
SCh_Distr consumes the least power for 8 nodes. At this size, the static power overhead of the centralized arbiter is more significant than its energy savings due to smaller arbitration packets. Somewhat surprisingly, SCh_Distr also consumes less dynamic power than LumiNOC, although requiring two arbitration rounds to propagate information about both time slots and subchannels. This is due to multiple reasons: First, while both approaches require broadcasting, senders in SCh_Distr do not need to send the arbitration packets to themselves. Moreover, for synthetic traffic with just *one* packet size, the 'packet length' fields are not needed in both mechanisms, which leads to a smaller control packet size in SCh_Distr as this decreases the control packet to $(dst\_id + Src\_bitmap)$ (LumiNOC) and (Src_bitmap) (SCh_Distr). As these packets are broadcasted by each sender, the difference in control packet size becomes significant and outweighs the energy required for the unicast control packet of in phase 2. We aim to study the impact on dynamic power of two packet sizes (as common in multi-processors) on SCh_Distr in the future.

D. Discussion

Although subchannel scheduling requires more complex arbitration and more information to be exchanged prior to data transmission, both of our proposed schemes only incur small power and latency overheads for low network loads. In low-utilization scenarios, LumiNOC's speculative sending approach is superior since the arbitration latency becomes large compared to the actual data transmission, particularly for small packet sizes. For moderate and high utilization rates, both of our subchannel approaches are superior to LumiNOC both in terms of power, latency, and throughput.
For small number of nodes, distributed arbitration is the preferred design as the static power caused by the resource overheads of a centralized arbiter is more significant in these cases. For larger number of nodes, however, centralized arbitration is the preferred choice since arbitration packets are smaller which allows for significant dynamic power savings that justify the overheads of a separate arbitration circuitry. Besides, a centralized arbiter would facilitate the implementation of more sophisticated flow control mechanisms (e.g. requesting the bus for multiple packets), which is difficult to implement in distributed arbitration.
Our results confirm the effectiveness of scheduling senders in both time slots and subchannels, and that power overheads are negligible compared to the throughput gains. This can be leveraged to either provide higher performance to bandwidth-critical NoCs, or to enable low power designs since fewer number of buses are necessary to offer the same bandwidth to a NoC. As buses are typically the backbone of higher-order topologies in optical NoCs, we believe that these improvements would carry over to improve the overall power efficiency of a NoC implementing them. In addition, subchannel scheduling is not restricted to the on-chip domain and could likely improve link utilization of optical buses for inter-chip communication, too. For instance, recent proposals connecting multiple chiplets with optical buses to form a 'virtual chip' would also benefit from our proposal [20].

VI. Related Work

Bandwidth sharing techniques were shown to be an efficient architectural approach to decrease laser power by a number of studies, which all aim to keep arbitration overheads low and bandwidth utilization high [21], [22], [23], [24]. Some wavelength-routed ONoCs [9], [25] share bandwidth by sharing the address space to reduce the number of wavelengths in the network while providing collision-free routes. Contention resolution at the destination node is managed by a separate global control network on which senders perform destination-checking.
Other studies investigate token-based arbitration schemes: FlexiShare [26] is a channel sharing architecture that improves channel utilization; however, parallel wavelength channels are necessary for arbitration for both sender and receiver sides, causing additional costs in power and area. 'Channel Borrowing' [27] simplifies Flexishare's token arbitration scheme by restricting the number of senders on a shared channel to two, which leads to a more light-weight design. Corona [22] assigns dedicated channels to each node, but other nodes can compete for transmitting on other node's channels, too. These approaches report large utilization improvements, however, token-based arbitration is fundamentally limited in scalability as nodes have to wait in line until they receive a token.
'Wavelength Stealing' [28] is an arbitration-free channel sharing mechanism that uses erasure coding to recover from collisions. GASOLIN [29] implements an arbiter at each node to enable pipelined distributed global arbitration for MWMR crossbars. FeatherWeight [30] is a feedback-controlled arbitration scheme with QoS support through adaptive source throt-

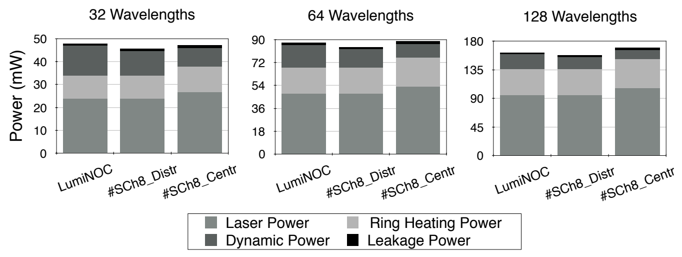


Fig. 10: Power Breakdown for 8 Nodes

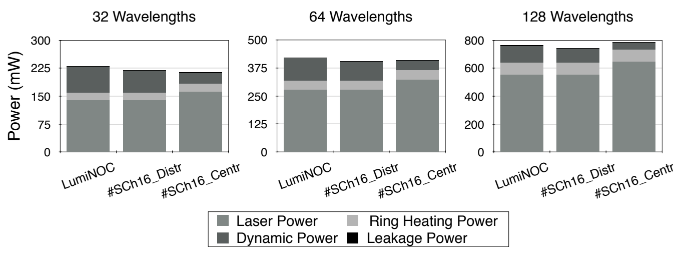


Fig. 11: Power Breakdown for 16 Nodes

tling. Although these approaches provide interesting studies to efficiently utilize bandwidth, they often apply them within the framework of a whole NoC design. Our proposed subchannel scheduling for shared buses offers higher flexibility as buses are a modular design that can be scaled conveniently both in size and bandwidth based on the system demands.

VII. Conclusion

This paper showed that scheduling transmissions on shared optical buses both in time slots and subchannels can be implemented efficiently by both distributed and centralized arbitration schemes and improves throughput tremendously. Power overheads of more complex subchannel arbitration are negligible, and latency is only degraded for small network loads. The higher the number of nodes connected to the bus and/or the higher the optical bandwidth on the bus, the more effective and superior our subchannel scheduling approach becomes. Implementing our approaches in buses that form the backbone of larger NoCs would provide higher bandwidth within the same power budget or could be leveraged to save power by lowering bandwidth without decreasing throughput compared to sequential-only scheduling.

VIII. Acknowledgements

This research was conducted with support from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671553 (ExaNeSt). Luján is supported by a Royal Society University Research Fellowship.

References

[1] Y. Demir and N. Hardavellas, "Ecolaser: an adaptive laser control for energy-efficient on-chip photonic interconnects," in *ISLPED*. ACM, 2014, pp. 3–8.
[2] C. Li *et al.*, "Luminoc: A power-efficient, high-performance, photonic network-on-chip," *TCAD-CEDA*, vol. 33, no. 6, pp. 826–838, 2014.
[3] R. Das *et al.*, "Catnap: energy proportional multiple network-on-chip," in *ACM SIGARCH Computer Architecture News*, vol. 41. ACM, 2013, pp. 320–331.
[4] S. Volos *et al.*, "Ccnoc: Specializing on-chip interconnects for energy efficiency in cache-coherent servers," in *NoCS'12*. IEEE, 2012, pp. 67–74.
[5] K. Bergman *et al.*, *Photonic Network-on-Chip Design*. Springer, 2014.
[6] M. Georgas *et al.*, "Addressing link-level design tradeoffs for integrated photonic interconnects," in *CICC, 2011*. IEEE, 2011.
[7] M. Haurylau *et al.*, "On-chip optical interconnect roadmap: Challenges and critical directions," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 12, no. 6, pp. 1699–1705, 2006.
[8] B. Bohnenstiehl *et al.*, "A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *VLSI Circuits*. IEEE, 2016, pp. 1–2.
[9] P. K. Hamedani, N. E. Jerger, and S. Hessabi, "Qut: A low-power optical network-on-chip," in *NOCS, 2014*. IEEE, 2014, pp. 80–87.
[10] A. Shacham *et al.*, "Photonic networks-on-chip for future generations of chip multiprocessors," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1246–1260, 2008.
[11] E. Peter *et al.*, "Active microring based tunable optical power splitters," *Optics Communications*, vol. 359, pp. 311–315, 2016.
[12] J. Tang *et al.*, "High-speed tunable broadband microwave photonics phase shifter based on an active microring resonator," in *WOCC'16*. IEEE, 2016, pp. 1–3.
[13] D. M. Vantrease, "Optical tokens in many-core processors," Ph.D. dissertation, UNIVERSITY OF WISCONSIN–MADISON, 2010.
[14] C. Sun *et al.*, "Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *NOCS, 2012*. IEEE, 2012, pp. 201–210.
[15] C. Chen, J. L. Abellán, and A. Joshi, "Managing laser power in silicon-photonic noc through cache and noc reconfiguration," *TCAD-CEDA'15*, vol. 34, no. 6, pp. 972–985, 2015.
[16] M. J. Heck and J. E. Bowers, "Energy efficient and energy proportional optical interconnects for multi-core processors: Driving the need for on-chip sources," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 20, no. 4, pp. 1–12, 2014.
[17] Y. Ben-Itzhak *et al.*, "Hnocs: modular open-source simulator for heterogeneous nocs," in *SAMOS*. IEEE, 2012, pp. 51–57.
[18] K. Preston *et al.*, "Performance guidelines for wdm interconnects based on silicon microring resonators," in *CLEO: Science and Innovations*. Optical Society of America, 2011.
[19] S. Werner, J. Navaridas, and M. Lujan, *Designing Low-power, Low-latency Networks-on-chip by Optimally Combining Electrical and Optical Links*. IEEE, 10 2016.
[20] Y. Demir *et al.*, "Galaxy: A high-performance energy-efficient multi-chip architecture using photonic interconnects," in *ICS'14*. ACM, 2014, pp. 303–312.
[21] N. Kirman and J. F. Martínez, "A power-efficient all-optical on-chip interconnect using wavelength-based oblivious routing," in *ACM Sigplan Notices*, vol. 45. ACM, 2010, pp. 15–28.
[22] D. Vantrease *et al.*, "Corona: System implications of emerging nanophotonic technology," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 153–164.
[23] P. Koka *et al.*, "Silicon-photonic network architectures for scalable, power-efficient multi-chip systems," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 117–128.
[24] Y.-H. Kao and H. J. Chao, "Blocon: A bufferless photonic clos network-on-chip architecture," in *NOCS'11*. IEEE, 2011, pp. 81–88.
[25] S. Werner, J. Navaridas, and M. Luján, "Amon: An advanced mesh-like optical noc," in *HOTI'15*. IEEE, 2015, pp. 52–59.
[26] Y. Pan, J. Kim, and G. Memik, "Flexishare: Channel sharing for an energy-efficient nanophotonic crossbar," in *HPCA'10*. IEEE, 2010, pp. 1–12.
[27] Y. Xu, J. Yang, and R. Melhem, "Channel borrowing: an energy-efficient nanophotonic crossbar architecture with light-weight arbitration," in *ICS'12*. ACM, 2012, pp. 133–142.
[28] A. Zulfiqar *et al.*, "Wavelength stealing: an opportunistic approach to channel sharing in multi-chip photonic interconnects," in *MICRO'13*. ACM, 2013, pp. 222–233.
[29] J. Liu, J. Yang, and R. Melhem, "Gasolin: Global arbitration for streams of data in optical links," in *IPDPS'15*. IEEE, 2015, pp. 93–102.
[30] Y. Pan, J. Kim, and G. Memik, "Featherweight: low-cost optical arbitration with qos support," in *MICRO'11*. ACM, 2011, pp. 105–116.