Ziyue Zhang, Didier Colle, Wouter Tavernier, and Mario Pickavet, "On the network design and control of an optical network: interconnecting multiple chips on a wafer," J. Opt. Commun. Netw. 15, 119-132 (2023). Final version DOI:
https://doi.org/10.1364/JOCN.474187

On the network design and control of optical network – interconnecting multiple chips on wafer

ZIYUE ZHANG[1,*], DIDIER COLLE[2], WOUTER TAVERNIER[2], AND MARIO PICKAVET[1]

[1] *Department of Information Technology, Gent University - IMEC, Belgium*
[1] *IMEC - Department of Information Technology, Gent University, Belgium*
[*] *Ziyue.Zhang@UGent.be*

Compiled January 3, 2023

In this paper we propose a new network architecture for multi-chip optical Network-on-Wafer(NoW), we concentrate on the research of its control mechanisms and control algorithms. Our proposed optical NoW aims at providing dynamically controlled Tera-Bytes-per-second unidirectional bandwidth for every chip module in a multi-chip processor. This architecture is promising in achieving low energy consumption and high aggregated bandwidth, providing a competitive idea for the next generation of optical-connected multi-chip computing systems. A synchronous network control scheme with a network control algorithm is proposed for slow-varying traffic patterns. Moreover, edge coloring algorithm is an important part of our network control algorithm, we propose improved edge coloring algorithms which are modified from the existing edge coloring algorithms. We show that our improved edge coloring algorithm has lower time complexity, and it also achieves faster execution in our experiments than the existing methods.

http://dx.doi.org/10.1364/ao.XX.XXXXXX

1. INTRODUCTION

Modern processors tend to have multiple computing cores, in order to continue the growth of computational power on a single chip [1][2]. The size of a single chip continues to grow [3], however, the increasing number of cores and memory components on the chip brings dramatic increase in data communication traffic. Network-on-chip (NoC) technology has been established since the 2000s [4, 5] to overcome bottlenecks in chip-level communication. However, as the multi-core single chip scales in size, it is more prone to fabrication defects and thus has lower yield in fabrication.

A multi-chip processor interconnects multiple already tested chips to form a bigger processor on a wafer. It is a promising solution to build larger and faster processors with high yield. With the help of the novel 2.5-D integration technology [6][7], chips can be interposed on a wafer while an inter-chip network is fabricated on the wafer in order to accommodate the communication flows among chips. The inter-chip network is thus called Network-on-wafer(NoW) in this paper, while in some literature they are referred to as *inter-chip NoC*. Chips on the wafer can access shared memory and execute instructions in parallel. This method is expected to continue the growth of computational power in one computing system despite the end of Moore's Law, and also greatly accelerate highly parallel computation tasks such as machine learning and climate simulations, etc. [8][9]

Modern multi-chip computing systems have employed electrical wires as interconnects[10–13]. The inter-chip electrical wires bring mainly two problems. One is the high signal delay, the telegrapher's equation describes the signal propagation velocity v depending on resistance R, capacity C, and induction L in an electronic circuit:

$$\frac{\partial^2 v}{\partial x^2} = RC\frac{\partial v}{\partial t} + LC\frac{\partial^2 v}{\partial t^2} \quad \textbf{(1)}$$

For instance, the delay of a $10mm$ long on-chip electrical wire can already be a few hundreds of pico-seconds[5], which is close to one clock cycle in a processor. Another problem is that wafer-scale electronic interconnects consume very high energy. For example, power consumption of an intra-chip electronic connection ($\approx 1mm$) is about $0.1\ to\ 0.2pJ/bit$, while inter-chip electrical wires (a few hundreds of millimeters) can take up to $30pJ/bit$ [14]. The energy consumption of inter-chip electrical wires is a huge overhead compared with that of the floating-point operations in a modern processor ($\approx 1.7pJ/bit$) [15].

As a promising solution for implementing the next generation inter-chip network, silicon photonics technology can improve both on signal delay and power consumption. Light signals traverse waveguides with the speed of light $v = \frac{c}{n_{eff}}$, where $c \approx 3 * 10^8 m/s$ is the speed of light in vacuum and n_{eff} is the effective refractive index of waveguides. An integrated optical waveguide typically has $n_{eff} = 2 \sim 3$. For instance, the delay of a $10mm$ waveguide would be tens of pico-seconds. Moreover, energy consumption for sending optical signals can achieve $0.3pJ/bit$ [16] for both millimeter-level or sub-meter-level optical

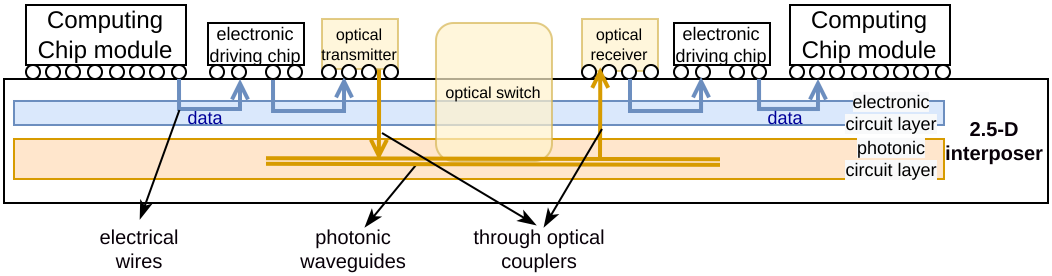

Fig. 1. Interposer composition

links thanks to modern optical transmitters and amplifiers based on III-V semiconductors working with high efficiency and low energy consumption [17–19]. Moreover, novel Silicon Nitride waveguides can achieve 3 to $10dB/m$ propagation loss [20]; ultra-low-loss waveguide crossings enable waveguides to be crossed within a low cost of attenuation and crosstalk [21]. They give flexibility to the design of photonic circuits which was impossible for the electronic ones. Silicon photonics waveguides also have the advantage that they guide simultaneously multiple optical signals due to Wavelength Division Multiplexing (WDM) technology.

It is evident that photonics technology is beneficial to be incorporated in the design of the next generation multi-chip NoW. An illustration of the concept of our proposed optical NoW architecture is shown in figure 1 and 2. Figure 1 depicts a vertical illustration of a 2.5-D interposer, the electronic wires and photonic waveguides are integrated in two layers. Figure 2 shows the idea of multi-chip processor composed of chip modules, i/o ports, electrical wires and optical NoW, etc.

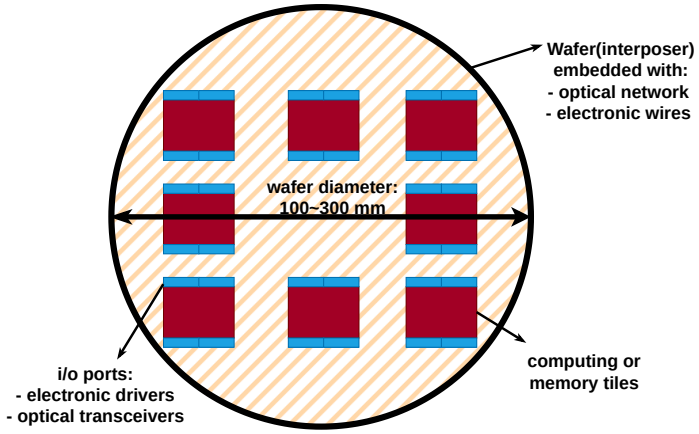

Fig. 2. Multi-chip processor on wafer

However, the control mechanisms used in traditional electrical NoC are not suitable for optical networks [22] due to the different working mechanisms of electrical routers and optical switches. Section 2 briefly reviews optical network control strategies in data center and photonic NoC networks, the unique challenges of photonic NoW will also be discussed. In section 3 we propose a new photonic NoW topology and a control scheme that is suitable for multi-chip computing systems. The network control algorithm designed for our NoW architecture will be shown in section 4. Section 5 will give details about our proposed edge coloring algorithms that were used in the network control algorithm in section 4. Section 6 concludes this paper and discusses the direction of future works.

2. RELATED WORKS AND THE NEW HORIZON OF NOW

Optical networks have already been widely deployed in data center and HPC (high-performance computing) networks due to the low power consumption and low packet latency offered by free-space optical devices, compared to the use of long electrical cables.

In data center and HPC networks, it is necessary to provide connectivity among a large number of network nodes with high bandwidth and low latency. However, the sheer number of nodes makes it infeasible to achieve all-to-all single-hop packet transportation. Researchers have therefore sought to minimize the number of hops traversed by packets through a combination of network topology design and control algorithms [23, 24].

Pre-designed cyclic switching methods have been used to control optical networks in data centers based on the assumption that uniform traffic patterns are common. These methods configure the optical switches in a cyclic fashion to ensure fairness among network nodes [25–27]. Another approach is to calculate and reconfigure the optical switches based on measured traffic demand, such as in the c-Throughput [28] and Helios [29] architectures.

In recent decades, the advancement of integrated photonics technology has also led to the consideration of deploying optical networks in NoC in academic research.

Guo et.al. [30] reported an integrated optical network architecture where optical waveguides replace electrical wires. The power consumption overhead of Electrical-Optical-Electrical (E-O-E) conversions were detrimental, leading to hundreds of pico-Joule-per-bit energy consumption for the network [30]. Several new paradigms of optical NoC or NoW architectures have been proposed in the literature, thorough discussions can be found in book[22].

Network designs like [31, 32] employ a circuit-switching optical network and a packet-switching electrical network. The electrical network works as a traditional electrical NoC, it accommodates control messages in order to set up circuits in the optical network for transmitting payloads [31]. However, the latency and energy consumption of these electrical path-setup messages can contribute over 90% to the overall delay and power consumption when the payload packet sizes are as small as 256 Bytes [32]. The low efficiency makes this network design especially not suitable for multi-chip processors that tend to have very short packets (8 Byte and 136 Bytes) in steady states [33][34].

Some networks [35, 36] make use of optical buses (e.g., Multiple-Write-Multiple-Read (MWMR) optical bus) for data communication. These optical networks can implement network arbitration and allocation only using optical waveguides. How-

ever, the optical buses use only half of its time slots for payload and another half slots for arbitration [36], which leads to a 50% degradation of effective bandwidth.

A few networks like [15, 37] configure small-dimension optical switches according to a pre-designed cycled switching plan similar to the data center networks like [25–27]. They are good compromises if we do not have large-dimension switches for connecting all nodes in the network.

The integrated photonic network architectures mentioned above were mostly designed for NoCs, however, applying these network architectures directly in the design space of photonic NoW is not the best solution mainly for two reasons:

(1) Footprint restriction and radix of optical switches

The most apparent distinction between NoC and NoW is their different scales of footprints. NoC architectures need to suit in a chip ($10^2 mm^2$ to $10^3 mm^2$), in most cases small-radix (typically 4-by-4) optical switches are used. In contrast, NoW can be implemented on a wafer ($10^4 mm^2$ to $10^5 mm^2$). This allows large-radix optical switches and much larger photonic circuits to be incorporated in the network design of NoW, which were not possible in the photonic NoCs. For example, [38] has reported a 64-by-64 optical switch with $208 mm^2$ footprint, and a 128-by-128 optical switch with $272 mm^2$ footprint was reported in [39]. Large-radix optical switch is a key component of our proposed photonic NoW architecture, they are discuss further in section 3 A.

(2) Bandwidth demand

Another important trait of NoW is the high bandwidth demand among network nodes. 'Node' is defined in a network referring to sources or sinks of payload data. Therefore, one node in an NoC is defined by one processing core or one memory channel. Per-node bandwidth provided by an NoC should support inter-core and core-memory traffics. On the contrary, a node in an NoW represents a processing chip or a memory tile which may include hundreds of clusters of processing cores and memory channels. As a consequence, per-node bandwidth demand in an NoW is much higher than that of NoC. Leveraging multiple parallel high-radix switches in the NoW architecture can provide high per-node bandwidth, achieving low network diameter and simpler network topologies.

Electronic NoW with multiple parallel high radix routers has already been explored in the design of DGX-multiGPU[40] from NVIDIA. In our knowledge, we are the first to propose a photonic NoW architecture specifically designed for connecting multiple chips with multiple parallel large radix optical switches. However, the key difference between NVIDIA's electronic NoW and the photonic NoW is that the former relies on packet-switching routers, while our proposed photonic NoW is buffer-less and it adopts circuit switching. Our proposed photonic NoW will be introduced in section 3.

Pioneering research has shown that multi-chip computing systems can benefit from ultra-high inter-chip bandwidth. For instance, in [33] it has been shown that their simulated Multi-chip-module GPU system continues to have linear or even super-linear performance improvements for memory-intensive workloads when the inter-chip link bandwidth increases from $384\,GBps$ to $1.5\,TBps$. Most existing optical network architectures are not able to provide Tera-Bytes level point-to-point unidirectional bandwidths for multi-chip systems. The newest commercial electrical NoW fabric *NVswitch* provides $450\,GBps$ point-to-point unidirectional bandwidth [40], which is also far below our target.

Our proposal of photonic NoW aims at providing TBps-scale unidirectional bandwidth-per-chip in the multi-chip system. Although this high bandwidth is not yet feasible for the current chips in the market as far as the time that this paper is written, we believe this will be supported by the future chip designs as the off-chip bandwidth is a well acknowledged bottleneck for many benchmarks [33, 40, 41].

3. NETWORK DESIGN

Our proposed optical NoW architecture aims at providing Terabyte-per-second bandwidth for each network node, high bandwidth utilization and low control overhead. This work focuses on the topology and network control methods for the proposed architecture.

The network topology we propose is shown in figure 3. Our proposed architecture contains two networks, namely an optical payload network and a control network. The optical payload network contains computing or memory chips, optical switches and optical waveguides. The control network is used for controlling the bandwidth allocation in the optical payload network, it includes a network controller, chip-controller links and optical switch control wires. They will be explained in more details in the following subsections.

A. Optical payload network

T denotes the number of chips in the network, and S the number of optical switches. Each optical switch has T input ports and T output ports and connects to all the chips via optical waveguides. An optical switch establishes point-to-point connections from its input ports to output ports. Differences of our network design from the hybrid circuit-switching networks in [31, 32] are: (1) our control network has simpler topology. (2) our control network is only used for sampling traffic demands and updating routing tables periodically instead of reacting to payload packets frequently. (3) the packet always traverse single hop in our proposed network thanks to the large radix optical switches.

There are several options for physically implementing the optical switches, for examples:

(1) Micro-Electro-Mechanical-system(MEMs) optical switches normally have large port counts and are energy-saving. However, MEMs switches suffer from low switching speed. [39] has demonstrated a 128×128 MEMs switches with about $1\,\mu s$ switching time.

(2) Mach-Zehnder-Interferometer(MZI)-based optical switches are fast in switching, but the port counts will be lower due to attenuation and crosstalk limitations. A 64×64 integrated MZI optical switch has been demonstrated in [38] with about $10\,ns$ switching time. It has also been shown in [42] that by using push-pull over-coupling Micro-Rings, MZI switches are capable of wavelength-routing, i.e., every wavelength is routed independently. The possibility of wavelength-routing greatly improves the granularity of bandwidth allocation in the network.

(3) Optical MWMR buses can serve as crossbars in the frequency domain with high port count. However, implementing optical MWMR buses need a huge amount of micro ring resonators which are temperature-sensitive in tuning. MWMR buses also require stable high power optical signal which may lead to high power consumption and wasted energy. MWMR buses can suit in our proposed network architecture but may lead to high implementation cost and implementation difficulty.

The photonic NoW architecture that we are proposing is theoretically compatible with any large radix optical switch.

There can be various trade-offs when choosing the technology of optical switches, due to their different switching times, implementation costs, abilities of wavelength routing, etc. However, analyzing the trade-offs of physical layer implementation falls out of the scope of this paper.

To be concrete we assume $T = 16$, i.e., the same number of GPU tiles in the Gen-3 NVSwitch multi-chip GPU system [40]. We will also assume that the switches are 16×16 wavelength-routing integrated MZI switches. Assume the number of wavelengths in the WDM system is 16, and each WDM wavelength channel operates at $25\,Gbps$ default bit rate, each MZI switch can provide $16 \times 25\,Gbps = 50\,GBps$ bandwidth from any input port to any output port. If we take for instance $S = 32$, the unidirectional aggregated bandwidth owned by each chip is $S * 50\,GBps = 1.6\,TBps$, which is a feasible solution in our target zone.

There are four types of optical payload packets being sent in the payload network: read request ($8\,Bytes$), read reply ($136\,Bytes$), write request ($136\,Bytes$) and write reply ($8\,Bytes$). These packets will be sent via source chips' output port and then traverse optical waveguides and optical switches, finally arriving at the destination chips' input port. An input port of a chip consists of a WDM de-multiplexer and a group of receivers working at each wavelength, and each output port of a chip consists of a WDM multiplexer and a group of transmitters at each wavelength.

The packets will only traverse single hop in the network. In order to send the packets to the correct destination chip, every source chip needs to know which receiver is being connected to its transmitters. This information should be stored in routing tables inside every chip, and the routing tables will be updated according to the decision of the network controller.

B. Control network

The network controller takes the responsibility of allocating optical channels to source-destination (s-d) pairs. The network controller can be implemented as Field Programmable Gate Arrays (FPGAs) or a CPU. The role of the network controller can also be taken by the server[1] of the multi-chip computing system. The network controller needs to execute network control algorithms in order to decide how to allocate available optical channels, it also needs to obtain inputs for this control algorithm from the chips and then send the output of this control algorithm to the optical switches and the chips. We propose such a control algorithm in section 4. The network controller and the optical switches have a master-slave relationship, the instructions of switch reconfiguration are sent via electrical wires. The chip-controller links can be implemented with optical waveguides or commercial high-speed electrical links like PCI-e buses[12].

There will be three types of packets sent in the control network:

(1) from chips to network controller: bandwidth demands.

The methodology is trying to predict bandwidth demands in the next period of time based on the traffic statistics measured in the last sampling period. Every chip generates traffic statistics by keeping track of the number of packets that has been generated during the last sampling period. These historical packets are categorized according to their destination chip id and contribute to the traffic statistics stored in the source chip. The traffic statistics will be translated into bandwidth demands and sent to the network controller. The bandwidth demand can simply be calculated as the amount of data divided by the sampling time and divided by the bandwidth of one channel. Denote $d_{i,j}$ as the bandwidth demand from chip i to j. Denote the amount of data generated during a sampling period that is from chip i to j as $\epsilon_{i,j}$ $[Bytes]$, sampling period is T_{sample} $[s]$.

$$d_{i,j} = \lceil \frac{\epsilon_{i,j}}{T_{sample} * B_{channel}} \rceil \quad \textbf{(2)}$$

Define one *channel* as the granularity of bandwidth allocation, thus one *channel* corresponds to a $B_{channel} = 25Gbps$ point-to-point connection in a wavelength-routing switch. The unit of $d_{i,j}$ is [number of *channels*]. With $T = 16$ and $S = 32$, the size of this traffic demand message is $15 + 2 = 17\,Bytes$, taking $1\,Byte$ for each integer demand value ($T - 1$ integer demand values, one per destination chip id) and $2\,Bytes$ for header and tail.

(2) from network controller to chips: routing table updates.

After receiving the estimated bandwidth demand messages from the chips, the network controller calculates how to allocate bandwidth resources to match with the demands all over the network. The result will be sent to the chips for updating their routing tables. With $T = 16$ and $S * W = 32 * 16 = 512$, the size of each routing table update message should be $256 + 2 = 258\,Bytes$, taking $4\,bits$ for indicating the destination chip id for each transmitter in the source chip, and $2\,Bytes$ for header and tail.

(3) from network controller to optical switches: reconfiguration.

The results of the network control algorithm will also be sent to the optical switches to give instructions of switch reconfiguration.

There can be different strategies for allocating the bandwidth resources in the network, we propose a synchronous network control scheme. This working scheme will need the network controller to execute the control algorithm periodically in time and concurrently inform the chips and optical switches about the algorithm results. The routing tables in the chips and the configuration in the optical switches should update according to this algorithm's result synchronously. The synchronous network control algorithm shall be executed depending on the significance of demand variation (see line 4-6 in procedure *SNC(d)* of Algorithm 1, and discussions in section 4 B). Our network control strategy is similar to those used in the c-Throughput and Helios [28, 29] architectures, in that the optical switches are reconfigured in response to measured traffic demand while the network is running. The dynamic bandwidth allocation method has been shown to significantly reduce the completion time of the applications and offers significantly reduced latency in which the traffic demands between some hosts change slowly in the c-Throughput network.

In contrast to c-Throughput and Helios, our network control algorithm is more flexible and has lower time complexity, as we will demonstrate in Section 4. Moreover, our proposed photonic NoW network always provides single-hop packet transportation and our network control strategy aims to fully match bandwidth demand with allocation.

Study shows that in many scientific computing applications the bulk of inter-processor communication changes slowly (in order of seconds) [43]. Assume our proposed photonic NoW multi-chip system can be involved in a high performance computing machine. Depending on the problem sizes of the submitted jobs, new jobs are likely to be submitted on the time scale

[1] e.g., a cpu is called the server of a GPU system if this cpu assigns workloads to the GPU system

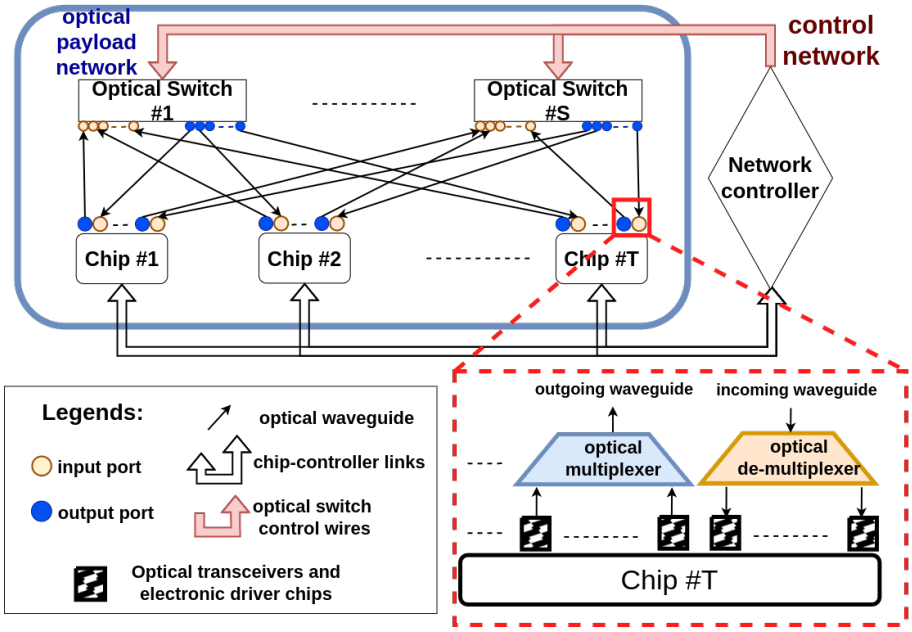

Fig. 3. Our proposed network Topology

of hours or minutes to the multi-chip system. Every job may launch tens to thousands or even more kernels concurrently or consequently, each kernel may last for seconds or minutes, or even hours. One kernel may have several steady states depending on its implementation, inter-chip traffic is assumed to be steady in these steady states in the kernels.

The execution time of the synchronous network control algorithm is critical for determining which scale of traffic pattern variations that this algorithm is capable of dealing with. The execution time of the network control algorithm should be at least one hundred times faster than the target scale of traffic pattern variation in order to react on time and avoid control overhead. In section 4 we propose such an algorithm. Further analysis of our proposed network control algorithm's execution time can be found in section 4 C.

C. Preventing buffer overflow

Buffer overflow can happen when a chip is receiving a payload packet and this chip cannot process this packet immediately nor has enough memory to store this packet, which will result into packet loss and may severely slow down the computation process. Credit-based flow control[44] has been widely used in traditional electrical NoCs to control traffic flows and prevent buffer overflow in electronic routers. Although electrical routers are no longer used in our optical payload network, similar credit-based flow control methods can be adopted in the network level.

At the start-up phase of the multi-chip system, each chip locally keeps an amount of credits for every other chip. A source chip sends payload packets to a destination chip, consuming a certain amount of credits. The source chip can no longer send payload to a destination chip when the credit is not enough but have to wait for the credit to increment.

Whenever a certain amount of received payload data has been processed (thus not occupying any buffer space) by a destination chip, it can rightfully increment the credit for the source chip. To update the credit in the source chip, we piggyback a 1 Byte integer data field for the accumulated credits in the next payload packet that will be sent from the destination chip to the source chip. In order to update credit information on-time, a threshold value of credit is set. If the number of credits need to be updated for another chip exceeds the threshold value, the chip generates a special optical packet in the payload network containing no payload data but only the credits to be updated.

4. SYNCHRONOUS NETWORK CONTROL ALGORITHM

The network controller executes the network control algorithm in order to allocate bandwidth resources to s-d pairs. We describe this network control algorithm in this section.

A. Problem description

The traffic demand message sent from a chip to the network controller contains $T-1$ integer values. The network controller receives T such messages, and thus they can be formulated into a $T \times T$ matrix with diagonal values all being zero. We denote this matrix as the "demand matrix" $\mathbf{d}$. Matrix $\mathbf{d}$ is the input of our network control algorithm.

The output of this algorithm contains the information of how to configure all the optical switches in the payload network. Every optical switch provides point-to-point connection and routes W number of wavelengths independently. Therefore, for every wavelength in an optical switch, the output of the algorithm can be seen as a $T \times T$ binary matrix with the summation of every row and column being at most 1, the rows in the matrix correspond to the ids of source chips and the columns destination chips. The diagonal elements are always 0 because a chip never sends packets to itself in this network. In total we have $S * W$ such binary matrices and they can be formulated into a 3-D binary allocation tensor $\mathbf{a}$. The three dimensions in tensor $\mathbf{a}$ correspond to the "wavelength id", "source chip id", and "destination chip id", respectively.

There are $S * W$ independent wavelengths in the network, they are assigned with unique wavelength ids. We give 'wavelength id no.1 to no.W' to the wavelengths in optical switch no.1,

and we give 'wavelength id no.(W+1) to no.(2*W)' to the wavelengths in optical switch no.2, etc. Note that here we give different wavelength ids to the same physical optical wavelengths in different optical switches because they are independent. An illustration of $\mathbf{a}$ is shown in figure 4 with $T = 3$. $a_{k,i,j} = 1$ means that wavelength k will be allocated to source chip i and destination chip j, otherwise not. The constraints on $\mathbf{a}$ are:

$$\begin{aligned} &\sum_{j=1}^{T} a_{k,i,j} \leq 1, \forall k, i \ , \ and \\ &\sum_{i=1}^{T} a_{k,i,j} \leq 1, \forall k, j \ , \ and \\ &a_{k,i,j} = 0 \ or \ 1, \forall i, j, k \ , \ and \\ &a_{k,i,i} = 0, \forall k, i. \\ with \ k \in \{1, 2, ..., S*W\}; &i, j \in \{1, 2, ..., T\} \end{aligned} \tag{3}$$

To calculate the allocation tensor a, a few simple algorithms have been proposed for similar network topology in the data center networks for the similar problems. The Hungarian algorithm [45] was employed to find maximum-weight matching repeatedly for each independent wavelength in [28] and [29]. This method has $O(SWT^3)$ time complexity in our application. This algorithm is greedy and not exact if we aim at fully matching demand and allocation, and it also has low potential to be parallelized because each execution of Hungarian algorithm needs the output of the result of the last execution of Hungarian algorithm.

We propose a new method aiming at fully matching demanded bandwidth and allocated bandwidth. This method first calculates matrix $\mathbf{a}'$ as an intermediate result, then we calculate tensor $\mathbf{a}$ from $\mathbf{a}'$ so that equation 4 is fulfilled.

$$a'_{i,j} = \sum_{k=1}^{S*W} a_{k,i,j}, \ \forall i, j \in \{1, 2, ..., T\} \tag{4}$$

Matrix elements $a'_{i,j}$ are integers and have the same unit as $d_{i,j}$: [number of *channels*]. The value of $a'_{i,j}$ means how many channels we would like to allocate from chip i to chip j. The objective is to match $\mathbf{a}'$ with $\mathbf{d}$ so that the channels are most utilized. We also would like $a'_{i,j}$ always being positive to avoid having high latency. In other words, $a'_{i,j} \geq 1$, we always keep at least one channel between any s-d pair despite the demand $d_{i,j}$ may be zero. $\mathbf{a}'$ thus has constraints in formula 5, the first two lines can be interpreted as each chip can only speak or listen maximally through $S * W$ number of channels.

$$\begin{aligned} &\sum_{i=1}^{T} a'_{i,j} \leq S*W, \forall j \ , \ and \\ &\sum_{j=1}^{T} a'_{i,j} \leq S*W, \forall i \ , \ and \\ &a'_{i,j} \geq 1, \forall i, j, i \neq j; \ \ a'_{i,i} = 0, \forall i. \\ with \ \ i, j \in \{1, 2, ..., T\} \end{aligned} \tag{5}$$

Tensor $\mathbf{a}$ can be calculated from matrix $\mathbf{a}'$ by using an edge coloring algorithm (see **Theorem 1**). The edge coloring algorithm sees every chip as a vertex in a bipartite multigraph and every *channel* as an edge. Different colors of the edges correspond to independent wavelength ids in tensor $\mathbf{a}$. The edge coloring algorithm can achieve $O(T^2 \lg(SW))$ time complexity[2] as will be shown in section 5. Every level of the recursion tree in the edge coloring algorithm is highly parallel, this means that our proposed algorithm has the potential to considerably speed up with multi-thread executions.

B. Algorithm

The pseudo-code of the proposed synchronous network control algorithm is shown in Algorithm 1. Procedure 1 'SNC' in Algorithm 1 is the main body of the network contorl algorithm, while the following three procedures (phase 1,2, and 3) are called inside procedure *SNC*. This control algorithm will be executed repeatedly while the network is running. The time interval between two executions is at least $\mathbf{T}_{control}$. Every $\mathbf{T}_{control}$ period of time, the network controller will receive a new demand matrix and compare it with the previous one. Lines 2-6 determine whether or not executing the rest of the algorithm by iterating over the demand matrix. The algorithm will continue by calling phase 1, phase 2, and phase 3 as long as one demand value has increased significantly ($d_{i,j} - (d_{prev})_{i,j} \geq d_{thre}$). Threshold d_{thre} works as a macro parameter in the algorithm for gauging the 'significance' of the demand increment. The value of d_{thre} should be set according to the bandwidth granularity of the network, for example it can be set as a few times of the lowest controllable bandwidth which is 25Gbps in our wavelength-routing assumption. Lines 2-6 have time complexity $O(T^2)$.

Lines 8-10 contain three phases of calculating the allocation tensor $\mathbf{a}$, they will be explained below.

• **Phase 1: proportional scaling**

First of all, in procedure *phase 1* of Algorithm 1 we calculate the sum of every row and column of the demand matrix, and keep the maximum summation value as MAX. Then we calculate the allocation matrix as a proportionally scaled demand matrix:

$$\begin{aligned} a'_{i,j} &= 1 + d_{i,j} * (S*W - (T-1))/MAX \\ &\forall i, j; \ \ i \neq j \end{aligned} \tag{6}$$

The first constant term in equation 6 is for allocating a full-mesh connection among chips, so that eventually $a'_{i,j} > 0$ for all i and j. This full-mesh connection consumes (T-1) independent wavelength ids, thus in the second term we subtract them from $S * W$ wavelength ids. The purpose of this step is to make sure every s-d pair always has at least one *channel* available in order to avoid high latency. Up to this point, at least one chip uses all its transmitters or receivers, i.e., $\exists j, \ \sum_{i=1}^{T} a_{i,j} = S*W, \ or, \ \exists i, \ \sum_{j=1}^{T} a_{i,j} = S*W$. We finally round the matrix elements with a floor function so that they are integers and constraints in formula 5 are all fulfilled.

phase 1 has time complexity $\Theta(T^2)$ by iterating over each matrix element three times.

• **Phase 2: allocating excess channels**

After phase 1 it is very likely that the constraints in formula 5 are not tight for all i and j, which means there will still be room for allocating the excess available *channels*. We assign excess *channels* in *phase 2* of Algorithm 1 according to a sorted queue of margins:

Define margin as equation 7.

$$m_{i,j} = \frac{a'_{i,j} - d_{i,j}}{d^{\alpha}_{i,j}} \tag{7}$$

[2] lg stands for logarithm with base 2 throughout this paper

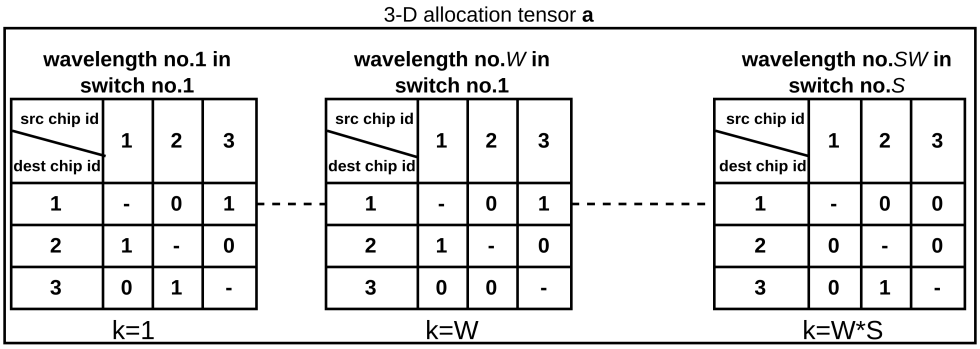

Fig. 4. An illustration of a 3-D allocation tensor **a**

Algorithm 1. Synchronous network control algorithm

```
1: procedure SNC(d)
2:     flag ← false
3:     for d_{i,j} in d do
4:         if d_{i,j} − (d_prev)_{i,j} ≥ d_thre then
5:             flag ← true
6:             break
7:     if flag is true then
8:         a′ = phase 1(d)
9:         a′ = phase 2(a′, d)
10:        a  = phase 3(a′)
11:        return a
```

```
1: procedure phase 1(d)
2:     MAX ← 0
3:     for all rows and columns in d do
4:         sum = SUM(row or column)
5:         MAX ← sum if MAX< sum
6:     for all (i, j) , i≠j do
7:         a′_{i,j} = 1 + ⌊d_{i,j} ∗ (SW − (T − 1))/MAX ⌋
8:     return a′
```

```
1: procedure phase 2(a′, d)
2:     create empty sorted queue Q_m
3:     Calculate summations of rows and column in matrix a′
       and store them in vectors sum_row and sum_col
4:     for all (i,j) do
5:         if (sum_row)_i < SW)&((sum_col)_j < SW)&(d_{i,j} >
       0) then
6:             insert m_{i,j} = (a′_{i,j} − d_{i,j}) / d^α_{i,j} in Q_m
7:     while Q_m not empty do
8:         m_{i,j} ← lowest margin in Q_m
9:         if ((sum_row)_i = SW)‖((sum_col)_j = SW) then
10:            delete m_{i,j} from Q_m
11:        else
12:            a′_{i,j} ← a′_{i,j} + 1
13:            (sum_row)_i = (sum_row)_i + 1
14:            (sum_col)_j = (sum_col)_j + 1
15:            update m_{i,j} = (a′_{i,j} − d_{i,j}) / d^α_{i,j} in Q_m
16:    return a′
```

```
1: procedure phase 3(a′)
2:     return Exact-edge-coloring(a′)
```

We calculate margin values and put them in a priority queue sorted in ascending order. Margin is defined in order to decide where to allocate excess bandwidth. When allocating a *channel* the algorithm starts with checking the lowest margin in the sorted queue Q_m, s-d pairs with lower margin values always have higher priorities in obtaining channel allocations. As a result, the difference between allocated bandwidths and bandwidth demands of all s-d pairs will be brought to a close number or numbers proportional to the bandwidth demands, depending on the value of parameter α:

When $\alpha = 0$, phase 2 aims at achieving a constant difference between the allocated bandwidth and the demanded bandwidth among all s-d pairs. When $\alpha = 1$, phase 2 aims at bringing differences between allocated bandwidth and demanded bandwidth proportional to the demanded bandwidth. Typically, the best scenario will be a mix of these two extreme cases, which means that the ideal value of α will be between 0 and 1, and can be tuned to adapt for different workloads of network.

It worth noting that the definition in equation 7 is not valid when $d_{i,j} = 0$ because the denominator would be zero. In this case, s-d pair (i, j) will not be inserted into the priority queue. However, this s-d pair will still be allocated with 1 *channel* because of equation 6.

In lines 4-6 in procedure *phase 2*, we only insert margin values in Q_m if incrementing $a_{i,j}$ by one does not violate constraints in formula 5. *phase 1* terminates when Q_m is empty, in other words, it terminates when no more *channels* can be allocated without violating the constraints.

Throughout the algorithm we keep track of the summation values for the rows and columns in matrix $\mathbf{a}'$ of vectors sum_row and sum_col. Assume the data structure of Q_m is a red-black tree, deleting or inserting a margin value in the tree has $\Theta(\lg T)$ time complexity, thus lines 3-6 have time complexity $O(T^2 \lg T)$. Line 10,12-15 each has time complexity $O(\lg T)$. Therefore, the **while** loop in lines 7-15 will eventually execute maximally SWT times for assigning every possible *channel* with $O(SWT \lg T)$ time complexity. In total the time complexity of phase 2 is $O(T^2 \lg T + SWT \lg T) = O(SWT \lg T)$, considering $SW >> T$.

• **Phase 3: edge coloring**

After phase 1 and 2, the allocation matrix $\mathbf{a}'$ is ready to be converted into the allocation tensor **a**. We show that this problem can be solved by exact edge coloring algorithms.

Theorem 1. If phase 3 adopts an exact edge coloring algorithm and matrix $\mathbf{a}'$ fulfills constraints in formula 5, tensor **a** can always be calculated from matrix $\mathbf{a}'$ while respecting the relationship in equation 4. The resulting tensor **a** fulfills constraints in formula 3.

Proof. Let **G** be a bipartite multigraph with $2 * T$ vertices. Δ denotes the maximum degree of vertices in **G**. There is no self-loop but there can be parallel edges in **G**. The bipartition of vertices is denoted as (A, B), where A and B are two vertex sets each having T vertices. Vertex set A and B represent all chips as sources and destinations of data in the payload network, respectively. Every edge in **G** connects one vertex in A and one in B, representing a *channel* allocated in the network. Constraints in formula 5 will be equivalent to constraint 8 on the maximum-degree in this multigraph if we take matrix $\mathbf{a}'$ as the adjacency matrix of multigraph **G**.

$$\Delta \leq S * W \tag{8}$$

In graph theory, an edge coloring of a graph is an assignment of "colors" to the edges of the graph so that no two edges incident on the same vertex have the same color. The edge coloring algorithm is equivalent in assigning independent wavelengths according to the allocation matrix in our problem. The equivalence comes from the fact that every independent wavelength in the switches provides point-to-point connections from transmitters to receivers.

It can be proved by induction [46] that the minimum required number of colors for coloring edges of a bipartite multigraph is exactly the maximum degree of this multigraph. Therefore, edge-coloring multigraph **G** with an exact algorithm yields a coloring with exactly Δ colors. This means that phase 3 will convert matrix $\mathbf{a}'$ to the configurations of Δ independent wavelength ids in **a**, if phase 3 adopts an exact edge coloring algorithm. Note that every color in the result of the edge coloring algorithm represents an unique wavelength id , i.e., the first dimension in tensor **a**. Because of inequality 8, theorem 1 is proved. □

Denote V and E as the number of vertices and edges in multigraph **G**, respectively. Denote Δ as the maximum degree of vertices in G. In our application $V = 2 * T$, $E \leq S * W * T$, and $\Delta = S * W$. Already known exact bipartite multigraph edge coloring algorithms have time complexity $O((E + V^2) \lg \Delta)$[47] and $O(E \lg \Delta)$[48, 49]. It will be shown in section 5 that we propose improved edge coloring algorithms with $O(V^2 \lg \Delta + E)$ time complexity, using our improved *Euler-division-improved* procedure in Algorithm 2 as subroutines.

• **Total time complexity**

In summary, time complexity of Algorithm 1 is $O(SWT \lg T + T^2 \lg(SW)) = O(SWT \lg T)$, dominated by phase 2. However, the time complexity expressions only show the worst cases, while the experiment results in section C will show that phase 3 actually dominates the execution time.

C. Algorithm execution time experiments

As discussed in section 3 B, the execution time of the network control algorithm is critical in response to the traffic pattern variation. In this section we roughly estimate the execution time of our proposed algorithm by showing our experiment results. We also analyze the execution time of each phase in our proposed algorithm with $T = 16$ and different values of $S * W$ so that we can expose the dominant phase for further optimization.

The edge coloring algorithms in phase 3 are implemented as the *Euler-color-improved* (algorithm 3) and *Gabow-color-improved* (algorithm 4) in figure 5 and figure 6, respectively.[3] As will be discussed in section 5, algorithm *Euler-color-improved* only guarantees to provide exact solutions when $S * W$ is a power of 2, therefore, we choose $S * W = 256$ and 512 in figure 5.

[3]These two algorithms are proposed later in section 5, proofs and time complexity will be shown.

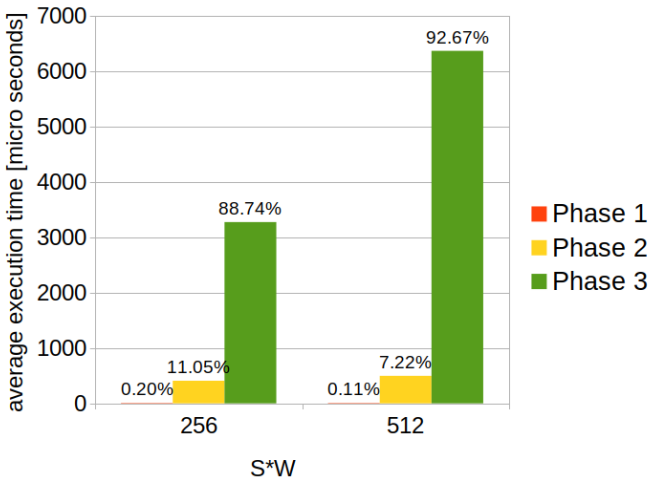

Fig. 5. Execution time composition of the network control algorithm, phase 3 is implemented as *Euler-color-improved* in algorithm 3

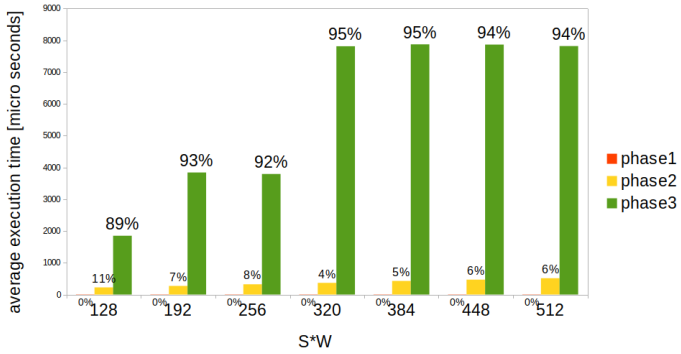

Fig. 6. Execution time composition of the network control algorithm, phase 3 is implemented as *Gabow-color-improved* in algorithm 4

The input demand matrices are randomly generated following independent uniform distributions in range $[0, 32]$ for every matrix entry. Every data point in figure 5 and figure 6 is averaged among 200 executions.

The experiments were run on a laptop Intel-i7 CPU with single-thread execution written in C++ language. In both cases phase 3 is the bottleneck that contributes about 90% to the total execution time even with our improved edge coloring algorithms. Although this leads to a different conclusion from the time complexity analysis in the previous subsection B, it is understandable because the time complexity analysis has only considered the worst cases and infinitely large input size for the algorithm.

In real life the execution of the network control algorithm can be affected greatly by many factors, such as:

• The capability (such as the number of cores and the core frequency) of the machine that runs the algorithm, the machine can be CPUs or FPGAs.

• The chosen programming language and the code optimization.

• The parallelism optimization of the algorithm. As will be shown in 5, the recursive edge coloring algorithm (i.e., phase 3 in algorithm 1) has potential to be implemented in multi-threads since all branches in the same level of the recursion tree are independent.

Other than the above factors, the algorithm itself can also be adjusted in order to fit in a shorter execution time. For example, enlarging the bandwidth allocation granularity from one wavelength channel (25GBps) to two wavelength channels (50GBps) can decrease the input size (the value of $S * W$) of the algorithm and thus approximately improve the algorithm execution time in half. This can be simply achieved by virtually bundling every two wavelength channels together in the logic of the algorithm, however, this will decrease the bandwidth control granularity and thus is possible to degrade the network performance.

In figure 5 and figure 6, millisecond scale execution time of our proposed algorithm has been reported, which is promising to respond to second scale traffic pattern variations. Although in reality ten or hundred times speed-up is possible to be achieved, it will then be possible to respond to sub-second scale traffic pattern variations.

In the next section 5, we discuss phase 3 of algorithm 1 in full details. And we will further explain how the execution time of phase 3 was improved compared with the existing methods.

5. IMPROVEMENTS ON PHASE 3

A. Preliminaries

Use the same notations for bipartite multigraphs in Theorem 1, denote the maximum degree of vertices in $\mathbf{G}$ as Δ, denote $e_{i,j}$ as an edge in $\mathbf{G}$ that incidents on vertex i and vertex j. An exact edge coloring algorithm should color the edges in $\mathbf{G}$ with exactly Δ colors. We virtually put two partitions of vertices in $\mathbf{G}$ on the left and right, namely the left vertex partition A and the right vertex partition B.

An Euler partition is a partition of the edges of a multigraph into open and closed paths, so that each vertex of odd degree is the end of exactly one open path, and each vertex of even degree is the end of no open paths. *Euler-division* procedure[4] was proposed in [50] to solve the bipartite edge coloring problem as a subroutine, the output of this procedure are two subgraphs of the input graph. At the beginning the two subgraphs have empty edge sets, *Euler-division* procedure then iterates over all the edges following the paths in the Euler partition of the input graph and puts them alternately into two sub-graphs $\mathbf{G}_1$ and $\mathbf{G}_2$. At the end of the procedure two sub-graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ should each contain half of the edges from the input graph. The maximum degrees of two subgraphs are either $\lfloor \frac{\Delta}{2} \rfloor$ *or* $\lceil \frac{\Delta}{2} \rceil$, and they can have same or different values. *Euler-division* procedure has time complexity $O(E+V) = O(E)$ for $E > V$ according to [50].

Pseudo-code and correctness of *Euler-division* procedure can be found in [50]. Representation of graphs in this procedure was assumed to be adjacency lists implemented with linked list data structure. Every edge $e_{i,j}$ needs to be present in both adjacency lists of vertex i and j. Time complexity $O(E+V)$ of *Euler-division* procedure can be achieved by keeping cross-references between $e_{i,j}$ in vertex i's adjacency list and that in vertex j's adjacency list. In this way, accessing the first element in vertex i's adjacency list, deleting it from two adjacency lists by cross-reference, and inserting the edge into one of the two subgraphs take $O(1)$ time.

[4] it was called 'euler partition' in the original paper, here we changed the name to avoid confusion of terms

Euler-color algorithm is obtained by executing *Euler-division* procedures on the resulting subgraphs of the last executed *Euler-division* procedures [47, 50]. The edges in two resulting subgraphs of an *Euler-division* procedure are guaranteed not to be colored with the same color. In the end there are at most $2^{\lceil \lg \Delta \rceil}$ subgraphs of the original input graph, every subgraph has a maximum degree equals to 1, then we can color each subgraph with a unique color. *Euler-color* algorithm has $O(E \lg \Delta)$ time complexity. This algorithm is guaranteed to be exact only when Δ is a power of 2. One extra color (compared to the exact solution Δ) is induced if and only if the result of a call of *Euler-division* procedure satisfies equation 9. Δ_1 and Δ_2 denote the maximum degree of the resulting subgraphs of *Euler-division* procedure.

$$\Delta_1 = \Delta_2 = \lceil \frac{\Delta}{2} \rceil = \frac{\Delta+1}{2}, \quad \Delta \, is \, odd \tag{9}$$

Several exact algorithms have been proposed to avoid entering this case and thus always guarantee to provide exact solutions.

Cole's algorithm [48, 49] adds an additional step of finding a matching including all vertices with maximum degree if Δ is odd, which makes sure that the input graphs of *Euler-division* procedure always have even Δ. These algorithms have time complexity $O(E \lg \Delta)$ and $O(E(\lg \Delta + \lg V))$, respectively.

Gabow's algorithm [47] uses another approach for providing exact solutions: when case in equation 9 is entered, the algorithm uncolors edges in an arbitrary color and puts them back with all the uncolored edges and proceeds the recursion. Gabow's algorithm has time complexity $O((E+V^2)\lg \Delta + E)$ which equals to $O(E \lg \Delta)$ when $E > V^2$ and otherwise $O(V^2 \lg \Delta + E)$.

We now define another data structure **weighted-linked-list** for representing the adjacency lists of multigraphs. This data structure will be used in our algorithms later in this chapter. A **Weighted-linked-list** represents an adjacency list of a vertex in a graph. **Weighted-linked-list** is a doubly linked list, every entry contains three data fields: destination vertex's id of an edge, a cross-reference and the multiplicity of the corresponding degenerate edges (defined as the weight of this entry). Therefore in a **weighted-linked-list**, edges incident to the same destination vertex are kept by only one entry. V instances of such data structure are needed for representing a graph $\mathbf{G}$ with V vertices. Any edge in $\mathbf{G}$ must appear in two **weighted-linked-lists** concurrently, we keep their cross-references (i.e., the data address to its counterpart in another **weighted-linked-list**) in order to access each other in $O(1)$ time. All **weighted-linked-lists** of a graph $\mathbf{G}$ contain $O(min[E, V^2])$ entries in total.

B. Improving *Euler-division* procedure

We propose a variant of *Euler-division* procedure called *Euler-division-improved* procedure by incorporating **weighted-linked-list** for representing graphs. The key observation is that in *Euler-division* procedure if edge $e_{v,u}$ has multiplicity $g_{v,u}$, we eventually will add $\lfloor \frac{g_{v,u}}{2} \rfloor$ or $\lceil \frac{g_{v,u}}{2} \rceil$ degenerate edges $e_{v,u}$ into two subgraphs. Therefore, when the graph has edges with high multiplicity, it will save time to jump to the end of the above observation instead of iterating over all degenerate edges. When the multiplicity $g_{v,u} = 1$, *Euler-division-improved* procedure boils down to *Euler-division* procedure. As a result, *Euler-division-improved* procedure is as least as good as the original algorithm and will speed up when the multiplicity of any edge is bigger than 1.

Pseudo-code of *Euler-division-improved* procedure is shown in Algorithm 2.

Algorithm 2. Euler-division-improved

1: **procedure** EULER-DIVISION-IMPROVED(**G**)
2: Create subgraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ with the same vertex sets but empty edge sets
3: Create an empty queue S
4: Put (at the front of the queue) all vertices with odd degree into S
5: Put (at the end of the queue) all vertices with none-zero even degree into S
6: **while** S is not empty **do**
7: vertex $s \leftarrow$ the first (at the front of the queue) vertex in S
8: delete s from S
9: vertex $v \leftarrow s$
10: **while** $degree(v) \neq 0$ **do**
11: **if** v is in vertex set A **then**
12: $e_{v,u} \leftarrow$ an edge incident on vertices v and u
13: **if** $w_{v,u}$ is odd **then**
14: Add edge $e_{v,u}$ with weight $\frac{w_{v,u}+1}{2}$ into $\mathbf{G}_1$
15: Add edge $e_{v,u}$ with weight $\frac{w_{v,u}-1}{2}$ into $\mathbf{G}_2$
16: $v \leftarrow u$
17: **else**
18: Add edge $e_{v,u}$ with weight $\frac{w_{v,u}}{2}$ into $\mathbf{G}_1$
19: Add edge $e_{v,u}$ with weight $\frac{w_{v,u}}{2}$ into $\mathbf{G}_2$
20: Remove $e_{v,u}$ from **G**
21: **else** v is in vertex set B
22: (same as above, except when $w_{u,v}$ is odd we add edge $e_{u,v}$ with weight $\frac{w_{u,v}-1}{2}$ into $\mathbf{G}_1$ and $e_{u,v}$ with weight $\frac{w_{u,v}+1}{2}$ into $\mathbf{G}_2$
23: put (at the end of the queue) s to S if $degree(s) \neq 0$
24: **return** $\mathbf{G}_1$ and $\mathbf{G}_2$

Theorem 2. *Euler-division-improved* procedure provides equivalent results as *Euler-division* procedure.

Proof. Consider when the original procedure finds an edge $e_{v,u}$ in **G** with multiplicity $g_{v,u}$, it can iterate over all $g_{v,u}$ degenerate edges and alternately put them into two subgraphs. If $g_{v,u}$ is even, then exactly half of $g_{v,u}$ edges will be in both subgraphs. If $g_{v,u}$ is odd, there will be $\frac{g_{v,u}+1}{2}$ and $\frac{g_{v,u}-1}{2}$ edges in two subgraphs respectively.

Euler-division-improved procedure implements the consequence of the above observation instead of the process. When $w_{v,u} = g_{v,u}$ is even (lines 18-19 or in line 22), we directly add edge $e_{v,u}$ to both $\mathbf{G}_1$ and $\mathbf{G}_2$ with half of its weight and the algorithm continues without any risk of breaking the correctness.

When $w_{v,u}$ is odd (lines 14-16 or in line 22), if vertex v is in the left partition of graph **G**, we always give G_1 the extra edge. For the mirror case in line 22, if vertex v is in the right partition of graph **G**, we always give G_2 the extra edge. As a consequence, when the end vertex of the current Euler path shifts from the left(right) to right(left) partition, the next extra edge when an edge's weight is odd will be added to $G_2(G_1)$. This makes sure that the result is equivalent to that of *Euler-division* procedure. □

Theorem 3. *Euler-division-improved* procedure has time complexity $O(min[V^2, E+V])$.

Proof. Line 2-9 and line 23 are the same as that in the original *Euler-division* procedure but the time complexities are different due to different data structures. In the original procedure, this part of the code needs $O(E+V)$ time for calculating degrees of every vertex by iterating over all edges and putting the vertices into the queue S. However, in *Euler-division-improved* procedure we have $O(min[V^2, E])$ entries stored in the **weighted-linked-lists**. We then only need to calculate vertices' degrees by calculating the summation of weights of entries in **weighted-linked-lists** and thus it takes $O(min[V^2, E]+V) = O(min[V^2, E+V])$ time.

Line 10 takes $O(1)$ time by checking the **weighted-linked-list** of vertex v is empty or not. Line 12 accesses the first entry of vertex v's **weighted-linked-list** using $O(1)$ time. Line 13-19 add entries to the **weighted-linked-lists** of two subgraphs with $O(1)$ time. Line 20 deletes the entries in two **weighted-linked-lists** containing edge $e_{v,u}$ in graph **G**: it first takes $O(1)$ time to access and delete the first entry in vertex v's **weighted-linked-list**, and then we use the cross-reference to delete $e_{v,u}$ from vertex u's **weighted-linked-list** in $O(1)$ time. Therefore, lines 12, 13-20 each has time complexity $O(1)$. Eventually the while loop in line 10-22 will do the above operations to every entry of **weighted-linked-lists** in **G** exactly once and thus cost $O(min[V^2, E])$ time.

To summarize, *Euler-division-improved* procedure has time complexity $O(min[V^2, E+V])$. □

Our *Euler-division-improved* procedure has time complexity at least as good as the original procedure. When $E > V^2$ *Euler-division-improved* procedure has $O(V^2)$ time complexity. The more multiplicity edges have in **G**, the more speed-up our proposed procedure has over the original procedure.

C. Edge coloring algorithms

In this subsection we show that our improved procedure can be incorporated into edge coloring algorithms. The modified *Euler-division-improved* procedure only speeds up each level of recursion and has no influence on the recursion structure.

We improve *Euler-color* algorithm by incorporating *Euler-division-improved* procedure and using **weighted-linked-lists** for representing graphs. Pseudo-code of algorithm *Euler-color-improved* is shown in Algorithm 3.

Algorithm 3. Euler-color-improved

1: **procedure** EULER-COLOR-IMPROVED(**G**)
2: **if** $\Delta = 1$ **then**
3: color the edges in **G** with a new color
4: **else**
5: ($\mathbf{G}_1$, $\mathbf{G}_2$)=*Euler-division-improved* procedure(**G**)
6: *Euler-color-improved*($\mathbf{G}_1$)
7: *Euler-color-improved*($\mathbf{G}_2$)

The correctness of *Euler-color-improved* inherits from the original algorithm. Now we prove its new time complexity.

Theorem 4. *Euler-color-improved* algorithm has time complexity $O(E + min[V^2, E] \lg \Delta)$.

Proof. Line 3 will in the end color every edge $e_{i,j}$ in **G** exactly once, thus has time complexity $O(E)$. Define an entry of **weighted-linked-list** is *traversed* by *Euler-division-improved* procedure once if this entry appears in the input graph of a call of *Euler-division-improved* procedure. Line 5 will eventually *traverse* every entry multiple times in the **weighted-linked-lists** of **G** and create copies of every entry stored in the subgraphs. The

number of copies of an entry equals to its weight. Denote the number of times an entry in **weighted-linked-list** need to be *traversed* by line 5 summing over the recursion tree as $P(w)$, w is the weight of this entry. Then we have $P(w) = 2P(w/2) + 1$, thus $P(w) = \frac{1}{2} \lg w$. *Euler-division-improved* procedure spends $O(1)$ time on each entry it traverse each time. Because $w = O(\Delta)$ and there are $O(min[V^2, E])$ entries in all **weighted-linked-lists** of $\mathbf{G}$, line 5 totally takes $O(min[V^2, E] \lg \Delta)$ time.

Time complexity of *Euler-color-improved* algorithm equals to the total time complexity of line 3 plus that of line 5, thus equals to $O(E + min[V^2, E] \lg \Delta)$. □

When $E > V^2$, time complexity of *Euler-color-improved* is $O(E + V^2 \lg \Delta)$, otherwise $O(V^2 \lg \Delta)$.

Pseudo-code of *Gabow-color-improved* is shown in Algorithm 4, we incorporate data structure **weighted-linked-list** and *Euler-division-improved* procedure based on the original algorithm *Gabow-color*[47].

Algorithm 4. Gabow-color-improved

```
1:  procedure GABOW-COLOR-IMPROVED(G)
2:      if Δ = 1 then
3:          color the edges in G with a new color
4:      else
5:          (G1, G2)=Euler-division-improved procedure(G)
6:          Gabow-color-improved(G1)
7:          r = 2^⌈lg Δ/2⌉ − Δ2, where Δ2 is the maximum degree
    in G2
8:          remove edges of r colors from G1 and add them to
    G2
9:          Euler-color-improved(G2)
10:         if G is not Δ-colored then
11:             choose an arbitrary color α in G
12:             for all edges e in color α  do
13:                 uncolor e
14:                 augment(e)
```

The correctness of *Gabow-color-improved* inherits from the original algorithm. Now we prove its new time complexity.

Theorem 5. *Gabow-color-improved* algorithm has $O(E + min[V^2, E] \lg \Delta)$ time complexity.

Proof. If the algorithm enters line 5, the input graph will be split into two subgraphs by *Euler-division-improved* procedure. Each subgraph will either enter *Gabow-color-improved* in line 6 or *Euler-color-improved* in line 9. Denote each recursive call as a 'node' in the recursion tree. We first estimate time spent in *Gabow-color-improved* nodes and then the time in *Euler-color-improved* nodes.

Line 8 transfers $O(V * r)$ edges from $\mathbf{G}_1$ to $\mathbf{G}_2$. Summing over the recursion tree, the total number of edges being transferred is $O(E/2) + O(E/4) + O(E/8) + ... = O(E)$, these edges can be transferred with time complexity $O(V^2 \lg \Delta + E)$ by first iterating all entries of **weighted-linked-lists** in $\mathbf{G}_2$ every time line 8 is entered and keeping their addresses in a hash-table.

Moreover, for all *Gabow-color* nodes, line 5 has time complexity $O(min[V^2, E] \lg \Delta)$ in total. Lines 10-14 do $O(V)$ augments (see [47] for the *augment*), each augment takes $O(V)$ time. thus in total $O(V^2)$ time. Therefore, all *Gabow-color* nodes totally has $O(V^2 \lg \Delta + E + min[V^2, E] \lg \Delta) = O(V^2 \lg \Delta + E)$ time complexity.

Euler-coloring nodes have time complexity $O(min[V^2, E] \lg \Delta)$ in total. Therefore, all recursion nodes in *Gabow-color* together have $O(V^2 \lg \Delta + E)$ time complexity in total.

Line 3 will in the end color $O(E)$ edges, thus has time complexity $O(E)$. Therefore, time complexity of *Gabow-color-improved* is $O(E + V^2 \lg \Delta)$. □

Execution time experiments of the above edge coloring algorithms are shown in figure 7. Four algorithms are tested with $V = 2 * T = 2 * 16$: approximation algorithms *Euler-color* and *Euler-color-improved* , the exact algorithms *Gabow-color* and *Gabow-color-improved*. The maximum degree of the input graph varies from 250 to 600, corresponding to the number of independent wavelength ids in the allocation tensor. These algorithms are written in C++ language and run with single thread executions on a laptop CPU. Each data point is averaged among 200 executions with randomly generated input graphs. The execution time ranges between $AVE \pm MSE$ for each algorithm are shown in the colored zones in figure 7. We see significant execution time improvements achieved by our improved algorithms compared to the original algorithms. Algorithm *Gabow-color* and *Gabow-color-improved* approximately double in execution time whenever Δ exceeds a power-of-two value. When the maximum degree of input graph equals to 512, our algorithm *Euler-color-improved* has about 40% speed-up compared to *Euler-color*, algorithm *Gabow-color-improved* has about 20% speed-up compared to *Gabow-color*. Corresponding number of colors used by two approximation algorithms are shown in figure 8, which align well between the theoretical lower and upper bounds.

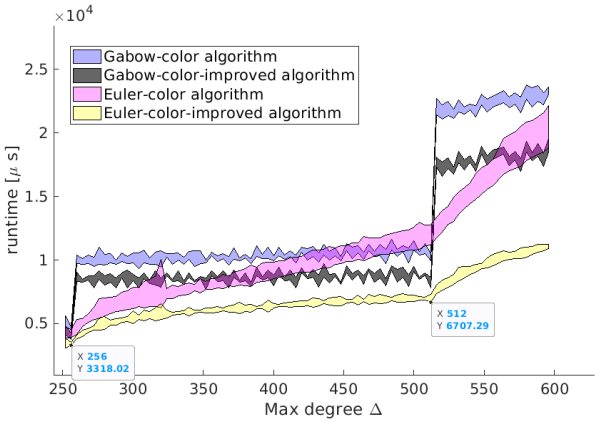

Fig. 7. Execution times of edge coloring algorithms

6. CONCLUSION

In this paper we have proposed a new optical NoW architecture aiming at Tera-Byte bandwidth for every chip in a multi-chip processor. This work has focused on the network control scheme and network control algorithms. Our synchronous network control scheme is flexible in bandwidth allocation, which dynamically allocates bandwidth resources for s-d pairs according to the measured bandwidth demands. A network control algorithm has been proposed for synchronous network control scheme. Our proposed architecture is promising in achieving low energy consumption and ultra-high bandwidth, which is suitable for the next generation of multi-chip processors.

We have also proposed an improved version of the *Euler-division* procedure with time complexity $O(min[V^2, E])$. As a

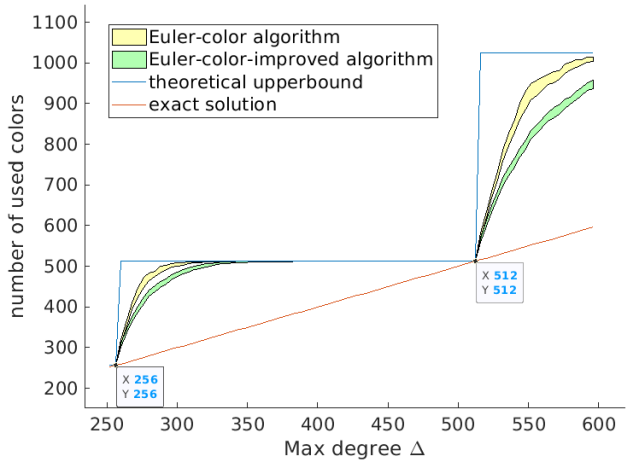

Fig. 8. Colors used by the approximation algorithms

result, the time complexity of bipartite edge coloring algorithm can achieve $O(E + V^2 \lg \Delta)$.

In future works faster heuristic or multi-thread algorithms for the edge coloring algorithm can be investigated for further improving the overall execution time of our synchronous network control algorithm.

Benchmark simulations with details of computing cores and memory blocks will be done in future works for further exploration of performance our proposed network architecture under real-workload traffics. We have planned to use a modified version of *booksim2*[51] as a standalone network simulator with synthetic traffic, we also plan to work with a modified version of *accel-sim*[52] as a full-system simulator for our proposed photonic NoW.

7. FUNDING AND ACKNOWLEDGMENTS

FUNDING

Part of this research is funded by Ghent University through the GOA-project 'Photonic Network-on-Wafer for Multi-Tile GPUs: From Architecture to Hardware Implementation' (01G01421).

The first author is supported by a PhD grant for strategic basic research of the Research Foundation – Flanders (FWO-V, 1S11323N)

ACKNOWLEDGMENTS

We want to thank our partner research groups in the GOA project for providing technological and industrial insights. We also want to thank FWO for funding this research.

The authors would like to thank OpenAI for the editing and proofreading support provided on January 3, 2023.

8. REFERENCES

REFERENCES

1. P. G. Howard, "Next generation intel microarchitecture nehalem," Tech. rep., Technical report, Microway Inc (2009).
2. S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain *et al.*, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," IEEE J. solid-state circuits **43**, 29–41 (2008).
3. C. H. Stapper, "On murphy's yield integral (ic manufacture)," IEEE transactions on semiconductor manufacturing **4**, 294–297 (1991).
4. A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Proceeding of the IEEE NorChip Conference,* (2000), 20, p. 0.
5. T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Comput. Surv. (CSUR) **38**, 1–es (2006).
6. K. Bernstein, P. Andry, J. Cann, P. Emma, D. Greenberg, W. Haensch, M. Ignatowski, S. Koester, J. Magerlein, R. Puri *et al.*, "Interconnects in the third dimension: Design challenges for 3d ics," in *2007 44th ACM/IEEE Design Automation Conference,* (IEEE, 2007), pp. 562–567.
7. A. Ivankovic, T. Buisson, S. Kumar, A. Pizzagalli, J. Azemar, and R. Beica, "2.5 d interposers and advanced organic substrates landscape: technology and market trends," in *International symposium on microelectronics,* (International Microelectronics Assembly and Packaging Society, 2015), 1, pp. 000041–000045.
8. K. Kurowski, M. Kulczewski, and M. Dobski, "Parallel and gpu based strategies for selected cfd and climate modeling models," in *Information Technologies in Environmental Engineering,* (Springer, 2011), pp. 735–747.
9. A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition,* (2016), pp. 4013–4021.
10. D. Foley and J. Danskin, "Ultra-performance pascal gpu and nvlink interconnect," IEEE Micro **37**, 7–17 (2017).
11. J. Nvidia, "Tesla k80 gpu accelerator," Board Specif. https://images.nvidia. com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf (2015).
12. D. D. Sharma, "Pci express® 6.0 specification at 64.0 gt/s with pam-4 signaling: a low latency, high bandwidth, high reliability and cost-effective interconnect," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI),* (2020), pp. 1–8.
13. J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," IEEE Micro **41**, 29–35 (2021).
14. S. Pasricha and M. Nikdast, "A survey of silicon photonics for energy-efficient manycore computing," IEEE Des. & Test **37**, 60–81 (2020).
15. C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications,* (IEEE, 2008), pp. 120–129.
16. H. M. Wassel, M. Tiwari, J. K. Valamehr, L. Theogarajan, J. Dionne, F. T. Chong, and T. Sherwood, "Towards chip-scale plasmonic interconnects," in *Workshop on the Interaction between Nanophotonic Devices and Systems (WINDS),* (2010), pp. 1–2.
17. J. A. Tatum, D. Gazula, L. A. Graham, J. K. Guenter, R. H. Johnson, J. King, C. Kocot, G. D. Landry, I. Lyubomirsky, A. N. MacInnes *et al.*, "Vcsel-based interconnects for current and future data centers," J. Light. Technol. **33**, 727–732 (2015).
18. G. Roelkens, L. Liu, D. Liang, R. Jones, A. Fang, B. Koch, and J. Bowers, "Iii-v/silicon photonics for on-chip and intra-chip optical interconnects," Laser & Photonics Rev. **4**, 751–779 (2010).
19. B. Haq, S. Kumari, K. Van Gasse, J. Zhang, A. Gocalinska, E. Pelucchi, B. Corbett, and G. Roelkens, "Micro-transfer-printed iii-v-on-silicon c-band semiconductor optical amplifiers," Laser & Photonics Rev. **14**, 1900364 (2020).
20. M. A. Tran, D. Huang, T. Komljenovic, J. Peters, A. Malik, and J. E. Bowers, "Ultra-low-loss silicon waveguides for heterogeneously integrated silicon/iii-v photonics," Appl. Sci. **8**, 1139 (2018).
21. Y. Ma, Y. Zhang, S. Yang, A. Novack, R. Ding, A. E.-J. Lim, G.-Q. Lo, T. Baehr-Jones, and M. Hochberg, "Ultralow loss single layer submicron silicon waveguide crossing for soi optical interconnect," Opt. express **21**, 29374–29382 (2013).
22. K. Bergman, L. P. Carloni, A. Biberman, J. Chan, and G. Hendry, *Photonic network-on-chip design* (Springer, 2014).
23. M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *SC'14: proceedings of the international conference for*

high performance computing, networking, storage and analysis, (IEEE, 2014), pp. 348–359.
24. W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20),* (2020), pp. 1–18.
25. M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM Conference,* (2016), pp. 216–229.
26. W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication,* (2017), pp. 267–280.
27. H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication,* (2020), pp. 782–797.
28. G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference,* (2010), pp. 327–338.
29. N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference,* (2010), pp. 339–350.
30. P. Guo, W. Hou, L. Guo, W. Sun, C. Liu, H. Bao, L. H. Duong, and W. Liu, "Fault-tolerant routing mechanism in 3d optical network-on-chip based on node reuse," IEEE Transactions on Parallel Distributed Syst. **31**, 547–564 (2019).
31. A. Shacham, K. Bergman, and L. P. Carloni, "Photonic networks-on-chip for future generations of chip multiprocessors," IEEE Transactions on Comput. **57**, 1246–1260 (2008).
32. E. Fusella and A. Cilardo, "H 2 onoc: A hybrid optical–electronic noc based on hybrid topology," IEEE Transactions on Very Large Scale Integration (VLSI) Syst. **25**, 330–343 (2016).
33. A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," ACM SIGARCH Comput. Archit. News **45**, 320–332 (2017).
34. A. Arunkumar, E. Bolotin, D. Nellans, and C.-J. Wu, "Understanding the future of energy efficiency in multi-module gpus," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA),* (IEEE, 2019), pp. 519–532.
35. D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System implications of emerging nanophotonic technology," ACM SIGARCH Comput. Archit. News **36**, 153–164 (2008).
36. S. V. R. Chittamuru, S. Desai, and S. Pasricha, "Swiftnoc: a reconfigurable silicon-photonic network with multicast-enabled channel sharing for multicore architectures," ACM J. on Emerg. Technol. Comput. Syst. (JETC) **13**, 1–27 (2017).
37. B. Zhang, H. Gu, K. Wang, Y. Yang, and W. Tan, "Low polling time tdm onoc with direction-based wavelength assignment," J. Opt. Commun. Netw. **9**, 479–488 (2017).
38. L. Qiao, W. Tang, and T. Chu, "Ultra-large-scale silicon optical switches," in *2016 IEEE 13th International Conference on Group IV Photonics (GFP),* (IEEE, 2016), pp. 1–2.
39. K. Kwon, T. J. Seok, J. Henriksson, J. Luo, L. Ochikubo, J. Jacobs, R. S. Muller, and M. C. Wu, "128$\times$ 128 silicon photonic mems switch with scalable row/column addressing," in *CLEO: Science and Innovations,* (Optical Society of America, 2018), pp. SF1A–4.
40. NVIDIA, "Nvidia http://https://www.nvidia.com/en-us/data-center/nvlink/," (2022).
41. A. Yazdanbakhsh, B. Thwaites, H. Esmaeilzadeh, G. Pekhimenko, O. Mutlu, and T. C. Mowry, "Mitigating the memory bottleneck with approximate load value prediction," IEEE Des. & Test **33**, 32–42 (2016).
42. Y. Huang, Q. Cheng, A. Rizzo, and K. Bergman, "Push—pull microring-assisted space-and-wavelength selective switch," Opt. Lett. **45**, 2696–2699 (2020).
43. K. J. Barker, A. Benner, R. Hoare, A. Hoisie, A. K. Jones, D. K. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld *et al.*, "On the feasibility of optical circuit switching for high performance computing systems," in *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing,* (IEEE, 2005), pp. 16–16.
44. W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks* (Elsevier, 2004).
45. H. W. Kuhn, "The hungarian method for the assignment problem," Nav. research logistics quarterly **2**, 83–97 (1955).
46. C. Berge, *Graphs and hypergraphs* (North-Holland Pub. Co., 1973).
47. H. N. Gabow and O. Kariv, "Algorithms for edge coloring bipartite graphs and multigraphs," SIAM journal on Comput. **11**, 117–129 (1982).
48. R. Cole, K. Ost, and S. Schirra, "Edge-coloring bipartite multigraphs in o (e logd) time," Combinatorica. **21**, 5–12 (2001).
49. R. Cole and J. Hopcroft, "On edge coloring bipartite graphs," SIAM J. on Comput. **11**, 540–546 (1982).
50. H. N. Gabow, "Using euler partitions to edge color bipartite multigraphs," Int. J. Comput. & Inf. Sci. **5**, 345–355 (1976).
51. N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS),* (IEEE, 2013), pp. 86–96.
52. M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA),* (IEEE, 2020), pp. 473–486.