Tuning Nanophotonic On-Chip Network Designs for Improving Memory Traffic

Yan Pan, John Kim[†], Gokhan Memik

Northwestern University
2145 Sheridan Road, Evanston, IL

[†]KAIST
Daejeon, Korea

ABSTRACT

The efficiency of on-chip network is important for future many-core processors, especially with the adoption of nanophotonics where the power consumption is dominated by static power. In this work, we study a recently proposed nanophotonic crossbar, FlexiShare, and explore techniques to further improve its performance, arbitration fairness and channel utilization under realistic memory related traffic load. We look at memory controller induced hotspot traffic and the bimodal packet sizes. Preliminary experimental results show that the baseline FlexiShare is capable of handling hot-spot traffic, but the token stream used in its global arbitration could be altered to improve arbitration fairness. We also identify the prevalence of bimodal packet sizes in real traffic load and propose four different schemes to enhance FlexiShare to efficiently support such traffic. Our results show that by adopting parallel networks of different datapath widths, we can reduce the workload execution time by up to 69% with the same cross section bandwidth. The pros and cons of each of the candidate schemes are also discussed.

I. INTRODUCTION

With technology scaling, more and more cores, together with other processing units (e.g., memory controllers) are integrated into a single chip. The large number of cores in future chip-multiprocessor (CMP) designs call for high performance on-chip networks, where data is routed in packets on shared channels instead of dedicated buses[1].Recent advances in nanophotonics make it an attractive alternative to conventional electrical signaling with its low latency and high bandwidth density [2]. Recently, an efficient nanophotonic crossbar, FlexiShare [3], was proposed to exploit the benefits of this emerging technology. FlexiShare advocates global sharing of the optical channel resources to minimize channel over-provisioning and employs a distributed token stream arbitration scheme for its global arbitration.

In this work, we take a closer look at FlexiShare and analyze its performance as well as possible architectural enhancements for better efficiency under two specific traffic scenarios: memory controller (MC) induced hot-spot traffic and bimodal packet-size traffic. The goal is to further improve the efficiency of the on-chip network under such realistic traffic load.

Even though on-chip bandwidth is likely to scale similar to the increase in the number of cores on a chip, off-chip bandwidth might not scale at the same rate. Hence, memory bandwidth is likely to become an important bottleneck in future processor architectures. At the same time, the pin number constraint will probably limit the number of memory controllers on-chip. Thus, a small number of MCs might be shared among a larger number of cores – resulting in hot-spot traffic pattern. The achieved bandwidth at the MCs may thus become the bottleneck for the overall performance of the processor. We note that only a few studies [4] have looked into the impact of MCs in an on-chip network and the available network designs are largely ignorant of the features of MCs. Here, we first evaluate a variation of the token stream arbitration in FlexiShare to see if it can improve the overall performance. Then, we adopt the same technique to address the fairness issue in FlexiShare when multiple MCs are placed in the network.

One important feature of real on-chip network traffic is that the packet size is bimodal, with long packets (64 or 128 bytes) carrying cachelines and short packets (8 to 16 bytes) containing control messages (e.g., memory requests, invalidations, etc.). Conventionally, such size difference is accounted for by splitting a long packet into multiple flits in an on-chip network. Prior works [5][2] on nanophotonic networks, including FlexiShare, largely assumed single-flit packets to leverage the high bandwidth density of nanophotonics. This incurs inefficiency when short packets are transmitted, especially considering the static power consumption of nanophotonic links [6]. Moreover, in the baseline FlexiShare, the token stream arbitration makes it difficult to hold on to a channel to transmit multiple flits in sequence. Additional re-order cost would be incurred if multiple packets (interleaved with other packets) are used to send a large chunk of data (e.g., a cacheline). In the second part of this paper, we explore different enhancements to the baseline FlexiShare architecture, in a bid to solve this problem. We describe each of the potential schemes, followed by discussion of their respective pros and cons.

The rest of the paper is organized as follows. In Section II, we briefly review the FlexiShare architecture and its arbitration schemes. We discuss the impact of MC-induced static hot-spot traffic pattern in FlexiShare in Section III. In Section IV, we turn our attention to bi-modal packet size support in FlexiShare and present various potential solutions. We summarize this paper in Section V and project future work.

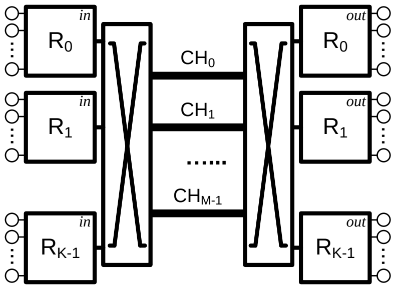

Figure 1.**Logical architecture of FlexiShare**

II. FLEXISHARE ARCHITECTURE

While nanophotonics enables repeater-less global communication and is ideal for implementing global crossbar network, it also incurs significant activity independent power consumption in laser and ring heating [6], which dominates the total power consumption of the links [3]. Thus, high utilization of nanophotonics link is needed to amortize the static power cost for an efficient design. However, conventional nanophotonic crossbar designs [5][7] dedicate channels to each router in the crossbar, hence requiring the number of optical channels to be proportional to the crossbar radix. When traffic is unbalanced (e.g., some nodes do not actively exchange data), the static power consumption in some of these dedicated channels are essentially wasted. To avoid such inefficiency, FlexiShare[3] was proposed to share a reduced number of optical channels across all nodes in the crossbar – hence detaching the channel provision from the network size and allowing high channel utilization to be achieved facing unbalanced traffic. An architectural diagram of FlexiShare is shown in Figure **1**. Concentration is assumed as each router is connected to multiple nodes. To illustrate the logical organization of the optical data channels (CH_i), we separate each router (R_i) into an input router (R_i^{in}) or the sending router and an output router (R_i^{out}) or the receiving router. However, they would be physically implemented as a single router. In FlexiShare, any router can send/receive on any channel and the number of channels (M) is independent of the crossbar radix (K) without sacrificing full connectivity.

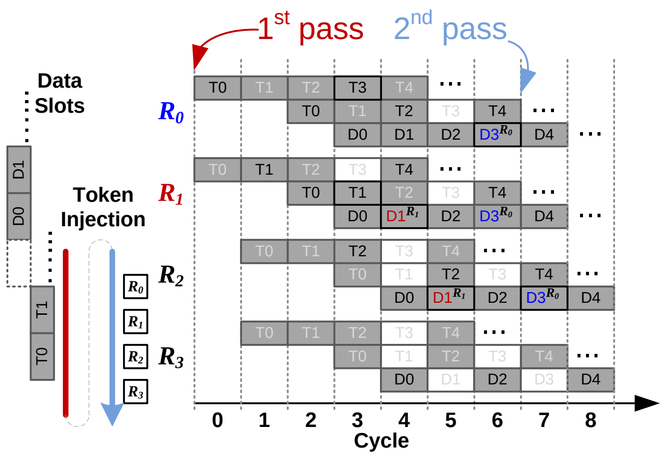

Figure 2.Timing diagram for a down-stream token stream for a radix-4 FlexiShare.

The sharing of the channels in FlexiShare is achieved by a token-stream arbitration scheme – to decide which router should occupy a specific channel. Conventional token-ring arbitration schemes, such as the one found in Corona [5], issue a single token for each channel across all the nodes and the node that grabs the token occupies the *whole channel*. With token-stream arbitration, however, a stream of tokens is continuously injected and the injected tokens flow across all the nodes. Each token represents the right to occupy the channel for a *time slot*, which is of the same length in time as the token. To avoid starvation, the token stream passes each router twice, as shown on the left of Figure 2. In the first pass, the tokens are dedicated. For example, in Figure 2, T_0 is dedicated to R_0, T_1 to R_1, T_2 to R_2 and T_3 to R_0 again[1]. This essentially implements a static dedicated time-slot scheme, where each node can occupy the channel for a fraction of the time. However, if the tokens come back to R_0 in the second pass without being grabbed, they are no longer dedicated and any node can grab any token in the second pass – implementing a daisy-chain-like priority scheme. Upon successfully grabbing a token, the router can then modulate its data onto the associated data slot (e.g., in Figure 2, R0 grabs token T3 and modulates its data onto data slot D3.), which comes several cycles after the token. Note that for efficiency, the token dedication information is not embedded in the token itself, but is maintained at each router using state machines.

III. MEMORY CONTROLLERS IN FLEXISHARE

Memory controllers in an on-chip network may incur hot-spot traffic pattern if they are shared among a larger number of cores. Hence, it is necessary to guarantee that the MCs are assigned a larger amount of bandwidth compared to a normal core.

A. *Repeated Token Scheme*

With the token-stream arbitration scheme of FlexiShare, the dedication of tokens represents the division of the bandwidth under heavy traffic load. Hence, it seems intuitive that we can dedicate more tokens for the MC nodes so that they get a larger share of the bandwidth. We devise a modified the token stream (repeated token scheme) such that the token for the MC nodes are repeated for ***n*** times while the cores' tokens are not increased. For example, the downstream token streams of the baseline and the repeated token scheme are as shown in Figure 3, assuming node 0 and node 8 are MCs.

B. *Experiment Setting*

To verify this hypothesis, we experiment with a 16-node radix-16 FlexiShare. To focus on the impact of the change in the global token arbitration scheme, we assume no concentration. Each router is connected to either one core or

[1] FlexiShare uses a pair of photonic links in opposite directions to implement a data channel. Thus, in the direction from R0 to R3, only R0, R1 and R2 are transmitting data and hence only these 3 nodes will be competing for tokens on the corresponding token stream. For the opposite direction, which is not shown in the figure, only R3, R2 and R1 will be competing for tokens on a separate token stream, which runs in the direction from R3 to R0.

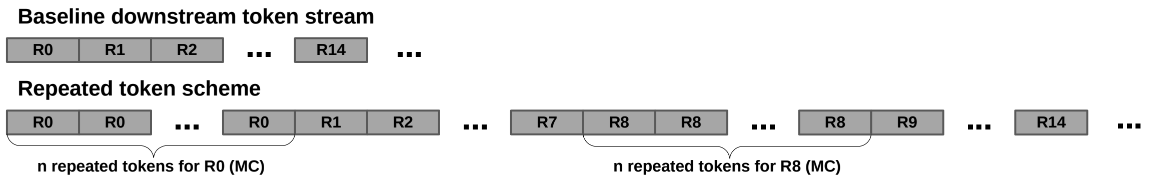
Baseline downstream token stream
R0
R1
R2
...
R14
...
Repeated token scheme
R0
R0
...
R0
R1
R2
...
R7
R8
R8
...
R8
R9
...
R14
...
n repeated tokens for R0 (MC)
n repeated tokens for R8 (MC)

Figure 3. Baseline token stream compared with the repeated token scheme (*n* tokens for each MC)

one MC. We assume 2 MC are connected in the network at node 0 and node 8 to create more conflicts and significant hot-spot traffic. We purposely choose two asymmetric locations for the MC nodes, for the fairness discussion in the coming subsections.

A synthetic workload is created where each of the cores in the network has a fixed amount of requests to send. Among these requests, ***x*** is MC request to one of the two MC nodes and the remaining ***(1-x)*** observes a uniform random (UR) distribution among all the nodes (cores or MCs) – in an effort to model cache coherence traffic among the cores. When ***x*** = 0, the workload is UR traffic, while when ***x***= 1.0, the workload is a pure hotspot traffic – i.e., all traffic are destined for two MC nodes. Upon receiving a request, the core or the MC will immediately generate a reply packet and send it back to the requesting node. Each core will keep sending out requests until its total number of outstanding requests reaches a threshold (16 in our experiment), at which point the core is blocked from sending more requests. The replies have a higher priority and are sent ahead of its own requests. We assumed single flit packets for both requests and replies. 8 channels are provisioned for the FlexiShare.

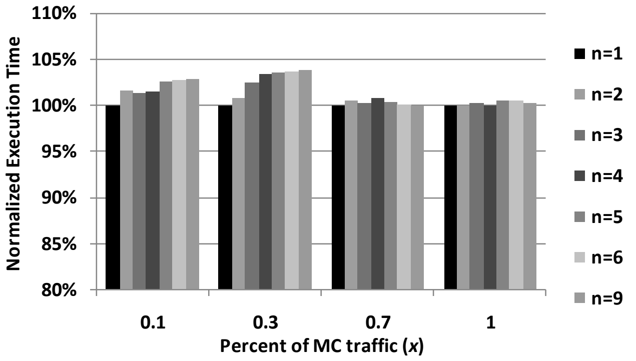

Figure 4.**Normalized execution time for various numbers of tokens for the MC nodes (n), normalized to the baseline token stream with (n=1).**

C. Performance Impact of Repeated Tokens

Figure **4** shows the normalized total execution time of the workloads. When ***n***=1, it is the baseline token stream [3] found in FlexiShare. In general, the increased number of tokens does not improve the performance and even slightly hurt the performance when MC traffic is low (***x*** ≤0.3). This strikes as surprising at first, as we intend to allocate more bandwidth to the MCs. However, after a careful analysis, the results turn out to be reasonable. First, the FlexiShare architecture inherently allocates more bandwidth to the more busy nodes by allowing any node to acquire remaining tokens on the second pass. When the MC nodes have high traffic (e.g., ***x*** = 1.0), all the core nodes are throttled by the maximum outstanding request limit and a new request can be sent only when the MCs finish sending a reply. Hence, the cores do not send as much requests any more, and the channels are not busy. For example, in the above experiment where ***x*** = 1.0, in any cycle, at most 2 requests (from the cores) and 2 replies (one from each MC) can be sent, while we have 8 channels (16 sub-channels in both directions) available for transmission. Thus, even with the baseline token stream, it is easy for the MC to capture unused tokens in the second pass and automatically occupy higher bandwidth. Note that allowing even higher number of outstanding requests will not improve the channel utilization as the bottleneck for throughput is with the MCs and not the round-trip latency of the request-reply pair.

At the same time, the repeated token scheme does have some negative effects. By dedicating more tokens to the MC nodes, it forces the non-MC nodes to rely more on second pass tokens (as they have less share in the first pass), and hence face more potential contention. For a radix-***k*** FlexiShare with ***m*** MCs, each having ***n*** repeated tokens, the non-MC nodes can get a first pass token only every ***(k-1+m*(n-1))*** cycles, compared to the ***(k-1)*** cycles in the baseline scheme. When the MC traffic is lower, the channel utilization is higher by the UR traffic, and the non-MC nodes will experience more rejections in the repeated token scheme – hence compromising the total execution time, as shown in the figure when ***x*** ≤ 0.3.

In addition, as shown in Figure 2, the delay between a first pass token and its associated data slot (e.g., first pass T0 is in cycle 0, D0 is in cycle 3, and the delay is 3 cycles) is longer than the delay between a second pass token and its associated data slot (e.g., 1 cycle between the second pass T0 and D0). Hence, repeating tokens for the MC essentially increases zero-load latency for packets sent from the MC nodes.

D. Fairness Impact of Repeated Tokens

While we might conclude that the baseline token stream is capable of handling hotspot traffic at the MCs, there is another problem. The two-pass token stream arbitration does not guarantee total fairness of the network [3]. For example, with the baseline token stream, node 0 is the first node on the second pass downstream token stream. Hence it can use up almost all the second pass tokens, while the other nodes (e.g., node 8) can only utilize its own dedicated tokens in the first pass in this direction. In the experiment described in Section III.A., assuming ***x***=0.3[2], we measured token request success rate of

[2] When the MC traffic load is dominant (x% =1), the channel utilization is very low. Thus, we assume lower MC traffic load here for a scenario with higher channel utilization.

both node 0 and node 8. For the baseline token stream, node 0 has a success rate of 77% while node 8's success rate is only 30%.This results in biased performance between the two MC nodes and increases performance variation.

To mitigate this problem, we can use the repeated token scheme to *only* increase the number of dedicated tokens for node 8. Figure **5**shows the impact of the number tokens for node 8 (***n'***) on the overall token request success rate of the two MC nodes. Note that all the other nodes, including the MC node 0, only have a single dedicated token. When ***n'=1***, it is the baseline token scheme. It is clear that by increasing the number of dedicated tokens to node 8, the gap of token request success rate between node 0 and node 8 can be significantly reduced and also minimize the variation between the two nodes.

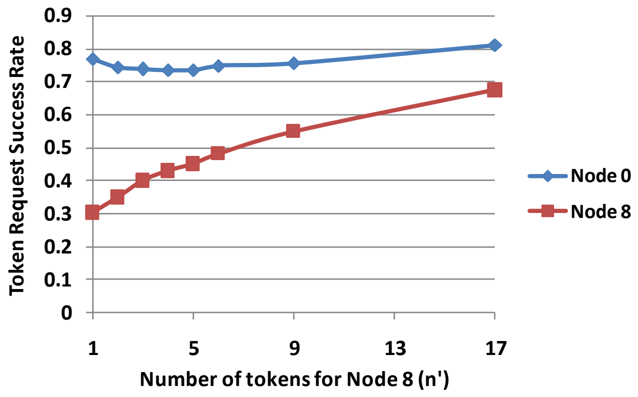

Figure 5.**Token-request success rate (8 channels, *x= 0.3*)**

For node 0, initially, the added dedicated tokens for node 8 slightly reduces its success rate, as more tokens are used by node 8 in the first pass. However, when more tokens are dedicated to node 8, they will not be fully utilized by node 8, and the unused tokens will come back in the second pass and benefit the success rate of node 0 again. This explains why the success rate of node 0 finally goes up with increase repeated tokens for node 8 in the figure. This also shows that by varying the token stream, it is not straightforward to eliminate the success rate gap between the two nodes. There could still be a gap even with ***n'*** = 17 dedicated tokens for node 8. This is because in this case, for a radix-***k*** FlexiShare, the ***n'*** repeated tokens only guarantees ***n'/(n'+k-2)*** of dedicated tokens for the node, and this is only 54.8% with ***n'=17***, ***k=16***. We may need a very large ***n'*** to bridge the gap, unless the success rate of node 0 is significantly hurt by the increased credits of node 8, which happens when the channels are more busy with traffic. For example, we also experimented with half the amount of channels (4 channels) in FlexiShare, as shown in Figure **6**. In this case, as the amount of channels is much fewer, the success rate of node 0 is significantly hurt initially and it is possible for node 8 to over-take node 0. But when more tokens are dedicated to node 8, they won't be fully utilized by node 8 and will end up benefiting node 0 again.

A side effect of the added repeated tokens is that they may also hurt the overall performance, similar to the discussion in the previous subsection (Section III.*C*.).

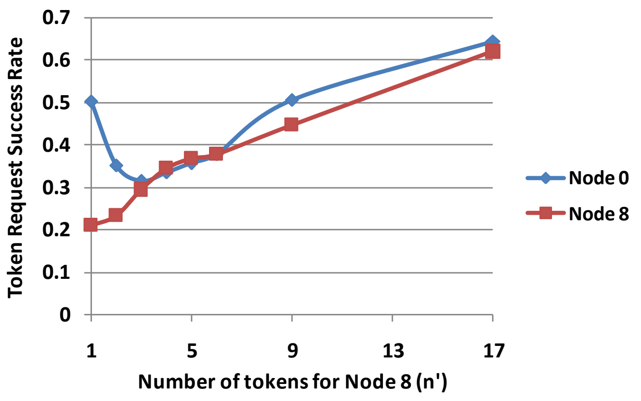

Figure 6.**Token-request success rate (4 channels, *x%=30%*)**

In summary, FlexiShare by nature is capable of handling hot-spot traffic pattern induced by MCs and can automatically allocate more bandwidth to the bandwidth-demanding nodes. However, the fairness of the bandwidth allocation among the nodes needs more research. While we may put 2 MCs at symmetric locations on a FlexiShare network (e.g., node 0 and node 15), the fairness study is nevertheless important when more MCs are present. We demonstrated that varying the token stream has some impact on the fairness issue.

IV. Supporting Bimodal Packet Size

A. *Break-down of on-chip traffic by packet size*

Real on-chip network load will be bimodal in packet size. The cacheline transfer constitutes long packets of over 64 bytes long; while other request, confirmation or invalidation messages are likely to be much shorter in the range of 8 to 16 bytes. To better understand the share of these two sizes of packets in an on-chip network, we analyze several network traffic traces generated using the SIMICS/GEMS simulator for a 64-node tiled CMP. In this simulator, the packets are explicitly bimodal, with short packets of 8 bytes and long packets of 72 bytes. Figure **7** shows the breakdown of *number* of packets in 9 SPLASH-2 and NU-MineBench benchmarks.

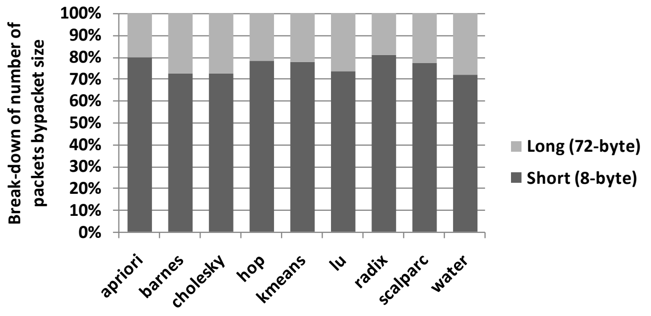

Figure 7.**Breakdown of total number of packets**

It can be seen that the number of short packets consistently constitute over 70% of the total number of packets. However, if we look at the bandwidth demand (or the total amount of data) break-down, the situation is the opposite – the long packets dominate, constituting mostly over 70% of the total data sent, as shown in Figure **8**.

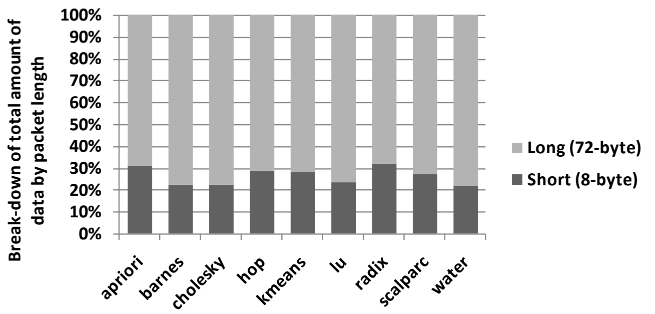

Figure 8.**Breakdown of total amount of data**

B. *Bimodal packets in On-Chip Network*

Conventionally, the different packet size in on-chip networks is accounted for by separating the long packets into multiple flits. However, if we choose a flit size the same as the long packets, we eliminate serialization latency [8] by having single-flit packets, but at the same time, a large portion of the channels will be wasted when sending short packets – resulting in poor channel bandwidth efficiency, especially when the number of short packets dominates. On the other hand, if we choose a flit size equal to the short packets, the long packets will experience long serialization latency (e.g., 8 cycles in our example) and as the amount of data in the long packets dominates, most of the traffic on the network is serialized.

The XShare [9] architecture aims to alleviate this problem by trying to combine two short packets to send on a wide data path in an electrical network. However, prior works [5][2] on nanophotonic on-chip networks, including FleixShare, have largely assumed single-flit packets with large flit size. This leverages the high bandwidth density provided by nanophotonics, but it also ignores the different packet sizes in on-chip network traffic and can result in inefficiency, especially considering that each nanophotonic channel consumes significant amount of static power in laser and ring heating [6]. Shacham et al.[10], took another approach to this problem and devised a circuit-switched nanophotonic network only for long packets (memory pages, cachelines, etc.) while sending the short packets in a parallel electrical network. However, in this case, the short packets will not be able to leverage the low latency of nanophotonics, resulting in long round-trip time for each request/reply pair.

C. *Enhancing FlexiShare for Bimodal Packet Sizes*

The FlexiShare architecture also assumed single-flit packets. However, different from previously proposed nanophotonic on-chip networks (e.g., Corona [5]) which can hold on to a channel and support multi-flit packets, the baseline token-stream arbitration scheme in FlexiShare makes it impossible to send multiple-flit packets in sequence. While it is possible to split the large chunk of data into multiple short packets and send them interleaved, the re-assembly and reordering needed at the receiver side will be costly, especially considering that most of the packets might need re-assembly based on results shown in Figure **8**. Thus, here we explore several possible enhancements to the FlexiShare architecture or its token-stream arbitration scheme to support bimodal packet sizes.

1) *Parallel Networks*

Balfour et al [11] proposed using two parallel networks of the same datapath width to improve channel utilization in an electrical concentrated mesh network. With the high bandwidth density in nanophotonics, we can extend this idea and devise two parallel FlexiShare networks, each with a datapath width matching the short or long packet size. Thus, all the packets in each network are still 1-flit long and no serialization latency is incurred for long packets. Whenever a router sends a packet, it will pick a network suitable for its packet size. Because FlexiShare can be provisioned with any number of channels [3], we can separately provision each of the parallel networks according to the average traffic load of that packet size. This works if the traffic load of each packet size is relatively stable over time. If the traffic load of each type alternates across time (e.g., with a phase full of short packets followed by a phase full of long packets), part of the channel resources will still be wasted – as this is a partially fixed channel dedication scheme and does not fully share the channel resources across all traffic types. However, depending on the length of the phases, adaptive schemes may be devised to turn on/off some of the channels in either network if its utilization is high/low for extended period of time – exploiting the fact that channel provision is independent from connectivity in FlexiShare.

We experiment with a 64-node, radix-16 FlexiShare network with the request/reply type of workload described in Section III.A., assuming $\boldsymbol{x} = 0$ and no MC node (i.e., a uniform random traffic without hot-spot). To show the impact of packet size variation, we make some of the packets 8 times as long as the others – forming a traffic load where part of the packets ($\boldsymbol{s}$) are short packets of 8-byte and the remaining packets ($\boldsymbol{1-s}$) are long packets of 64-byte. We provision the networks with a total of 512-byte cross-section bandwidth. This bandwidth is split, in different ratios, into a wide FlexiShare of 64-byte data path and a narrow FlexiShare of 8-byte data path. The total execution time of the workload is shown in Figure 9. In the figure, the legend W***a***_N***b*** (***t***) represents a network with ***a*** channels for the wide network, ***b*** channels for the narrow network and the total cross-section bandwidth is ***t*** bytes. Obviously, $\boldsymbol{t = a*64 + b*8}$. The baseline is a single FlexiShare network with only wide channels, i.e., W8_N0(512), and the total execution time of other schemes are normalized to that of the baseline. The x-axis shows different percentage of the number of short packets ($\boldsymbol{s}$). Comparing all the schemes with 512-byte cross-section bandwidth, it is obvious that dedicating some of the bandwidth to the narrow network can significantly increase the number of channels for short packets – resulting in higher overall throughput. This is illustrated by the consistent higher performance of W7_N8, W5_N24 and W4_N32 over the baseline W8_N0. When the percentage of number of short packets increases, intuitively it becomes favorable to provision more bandwidth to the narrow network.

With a fixed percentage of short packets, there is a sweet-point for the split-up of bandwidth between the two networks. For example, when $\boldsymbol{s}$ = 0.5 and 0.55, W7_N8

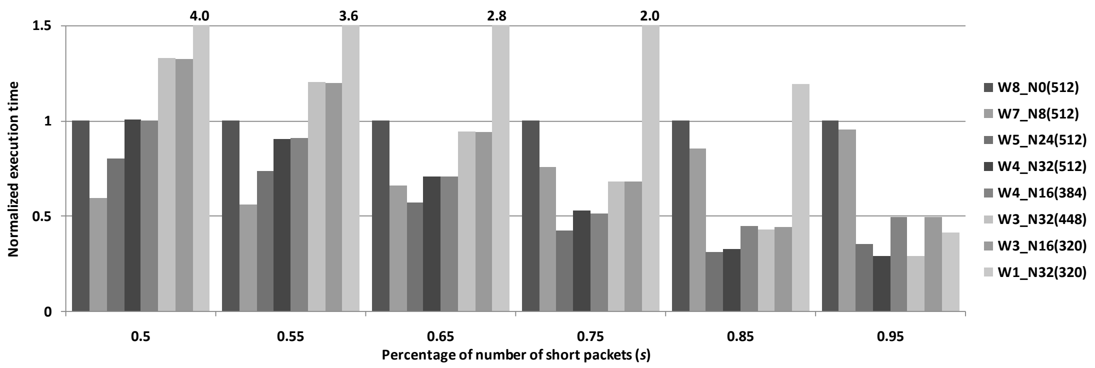

Figure 9.**Normalized execution time for various channel provisioning in parallel networks.**

performs best, reducing the total execution time by over 40%; while for ***s*** = 0.6 ~ 0.85, W5_N24 takes over as the most efficient, with a reduction in execution time as much as 69%.

Another point to be noted here is that excessive short channels are not necessary. For example, comparing W4_N32 and W4_N16, they performed similarly with ***s*** of up to 0.75, which means by employing two parallel networks, we can reduce the amount of channel provisioning by 25% without losing performance in these cases.

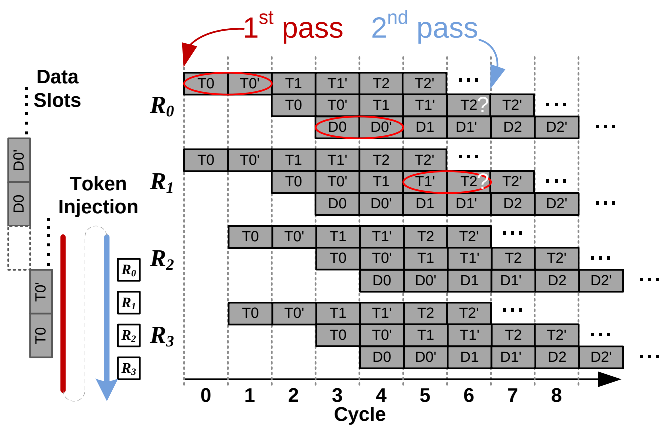

Figure 10**. Token Linkup within a single stream**

2) Distributed link-up of Tokens& Data slots.

Another natural extension of FlexiShare to support bimodal packet size is to use consecutive tokens / data slots for long packets so that we avoid the reassembly and re-ordering cost incurred by sending separate packets for a chunk of data. Similar to the baseline FlexiShare, this can be done in a distributed fashion. Instead of having the dedication of the tokens rotated every cycle, we can have consecutive tokens dedicated to a same router. For example, assuming the long packets are 2 flits long, for a radix-4 FlexiShare, in the downstream direction, we can have a token stream of ***T0***, ***T0’***, ***T1***, ***T1’***,***T2***,***T2’***…, as shown in Figure **10**. Here, token ***Tx*** and ***Tx’*** are both dedicated to router R_x.

To send single-flit packets, the routers will behave exactly the same as the baseline FlexiShare, trying to grab dedicated tokens (both ***Tx*** and ***Tx’*** for R_x) in the first pass, or any available token in the second pass.

Suppose R_0 is to send a long (2-flit) packet, it can grab token ***T0*** in the first pass in cycle 0 and it knows for sure the next cycle it has token ***T0’*** dedicated to it and it can send the two flits in sequence on data slots ***D0*** and ***D0’*** – essentially linking up the two tokens and data slots. However, if a long packet arrives at R_0 before cycle 1, it cannot grab ***T0’*** in the first pass in cycle 1 as ***T1*** in the next cycle is dedicated to R_1. In the second pass, tokens can also be linked up. For example, if a long packet arrives at R_1 before cycle 5, it can grab ***T1’*** in the second pass in cycle 5. However, there is no guarantee if R_1 can get ***T2*** in the next cycle. If the upstream R_0 grabbed ***T2***, R_1 will fail in getting ***T2*** and the granted ***T1’*** can no longer be used for the long packet. It can either be used for sending a short packet, if available, or it will be wasted. The link-up of the tokens and data slots can be easily indicated by a single bit in the head flit for the receiver to properly process the arriving flits.

The advantage of this scheme is multifold. First, we maintain the distributed arbitration and each router only relies on the token stream for global information. Second, there is no requirement for the number of channels to be provisioned. Even with a single channel, we can support the bimodal packet sizes and share all the channel resources across the network, without assumption for the nature of the traffic load. Third, the router design only needs slight modification from the baseline FlexiShare and the added complexity is moderate.

However, this scheme also has some problems. First, it faces fragmentation as in the second pass, there is no guarantee that two consecutive tokens can be linked up. In the worst case half of the bandwidth can be wasted (if alternative tokens are grabbed for short packets) while blocking the transmission of long packets. Second, the fragmentation problem aggravates with increased packet size – as the probability to successfully grab enough consecutive tokens on the second pass diminishes with increased number of tokens needed. Thus, this scheme might only work efficiently for long packets of no more than 2 flits. However, considering the large difference in size between large and short packets (8-byte vs. 64-byte), this is sub-optimal and will still waste part of the channel resource when sending short packets. Third, as the token grabbing for long packets on the second pass is speculative, it may result in the waste of tokens at upstream nodes while blocking even short packet traffic at downstream nodes.

3) Centralized link-up of Tokens and Data slots

To avoid the above mentioned fragmentation problem, another option is to have a centralized arbiter for sending long packets. When a node has short packets to send, it simply uses the baseline two-pass token arbitration scheme of FlexiShare.

However, if a long packet comes, it will first send a request, as a short packet, to a centralized arbiter, which makes an arrangement for the long packet on a certain channel for several slots in the future. Then the arbiter will send a short reply packet to the requesting node, informing it the channel and data slots allocated to it. The arbiter will also remove the corresponding tokens for those data slots from the token stream so that no other nodes can grab the tokens and occupy any of the granted data slots.

The advantage of this scheme is that we can have much narrower data path for short flits and more flits for long packets and we do not have to worry about fragmentation. However, there are also problems with this scheme. First, the communication with the centralized arbiter incurs extra zero-load latency. And such latency has to be carefully hidden within the router so as to avoid compromising throughput – resulting in significant router complexity. Second, for each long packet sent, 2 short packets are needed as overhead. Third, if narrow flit width is adopted and the long packets have many flits, serialization latency will be significant. Last, but not the least, the complexity of the centralized arbiter is high and it is not scalable with increased network size.

4) Channel Link-up

A complementary scheme to token link-up is channel link-up. With multiple channels present in the FlexiShare network, it is possible to linkup multiple channels to send a long packet in parallel. For example, Figure **11** shows the token streams of two channels at router R0. Here, ***Tx*** is dedicated to Rx in the first pass. We assume long packets are 2 flits long. In the first pass, if R0 has a long flit to send before cycle 0, it is guaranteed to get both ***T0***'s on the two channels and can send the two flits in parallel in cycle 3 on slot ***D0***'s. If a long packet arrives at R0 before cycle 4, however, it cannot grab the ***T1***'s as they are dedicated to R1, but it can try to grab the second pass tokens ***T2*** from the two channels. Similar to the token link-up scheme discussed previously, this scheme suffers from fragmentation and token wasting problems and it is also not easy to support many-flit packets. Compared to token link-up, this scheme might incur lower zero load latency as the long packets are sent in parallel. However, it also requires multiple channels to be provisioned.

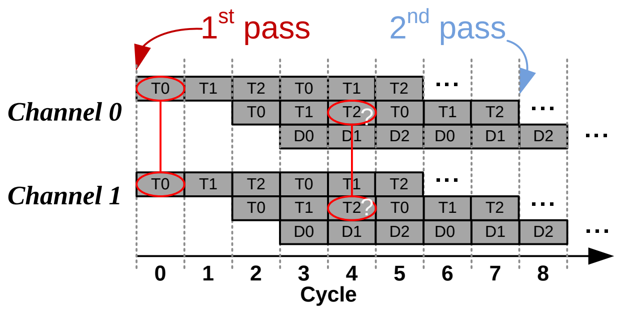

Figure 11**. Channel Link-up scheme at Router R0.**

V. SUMMARY AND FUTURE WORK

In this work, we focused on two aspects of on-chip network traffic : memory controller induced hotspot traffic pattern and bimodal packet sizes. With a small number of memory controllers shared among the cores, hot-spot traffic pattern can be formed. We found the baseline FlexiShare is capable of handling such hot-spot traffic thanks to its two-pass token stream arbitration. However, the baseline token stream needs to be altered to improve the fairness of the network.

To achieve better network efficiency, the bimodal packet distribution of on-chip network traffic needs to be exploited. We described four different schemes to enhance FlexiShare to support bimodal packet sizes. The link-up of tokens or channels fully shares the channel resources, but may not be efficient for very large multi-flit packet. Parallel networks of different datapath widths can be efficiently implemented and separately provisioned for high utilization rate. However, this requires a relatively stable mixture of the two types of packet lengths and may be inefficient if phases of a single type of traffic exist.

In the future, we plan to explore different token streams in FlexiShare to address the fairness issue and possibly employ adaptive schemes to enforce strict fairness. We will also further improve and evaluate the token/channel link-up schemes and try to minimize the fragmentation and token wasting problems under bimodal packet sizes. Extensive examination of more realistic on-chip network traffic will help understand which scheme is most efficient in a nanophotonic on-chip network.

REFERENCES

[1] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *Design Automation Conference, 2001. Proceedings*, 2001.

[2] C. Batten, A. Joshi, J. Orcutt, A. Khilo, B. Moss, C. Holzwarth, M. Popovic, H. Li, H. Smith, J. Hoyt, F. Kartner, R. Ram, V. Stojanovic, and K. Asanovic, "Building Manycore Processor-to-DRAM Networks with Monolithic Silicon Photonics," *2008 16th IEEE Symposium on High Performance Interconnects*, 2008, pp. 21-30.

[3] Y. Pan, J. Kim, and G. Memik, "FlexiShare: Energy-Efficient Nanophotonic Crossbar Architecture through Channel Sharing," *To Applear in Proc. of International Symposium on High-Performance Computer Architecture (HPCA)*, Bangalore, India: .

[4] D. Abts, N.D. Jerger, J. Kim, D. Gibson, and M.H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," *ACM SIGARCH Computer Architecture News*, 2009, p. 10.

[5] 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," *International Symposium on Computer Architecture*, 2008, p. 11.

[6] R.K. Dokania and A.B. Apsel, "Analysis of challenges for on-chip optical interconnects," *19th ACM Great Lakes symposium on VLSI*, 2009, p. 5.

[7] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary, "Firefly: illuminating future network-on-chip with nanophotonics," *Int'l Symposium on Computer Architecture*, vol. 37, 2009, p. 11.

[8] W.J. Dally and B.P. Towles, *Principles and Practices of Interconnection Networks (The Morgan Kaufmann Series in Computer Architecture and Design)*, Morgan Kaufmann, 2004.

[9] R. Das, S. Eachempati, A.K. Mishra, V. Narayanan, and C.R. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs," *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, 2009, pp. 175-186.

[10] A. Shacham, K. Bergman, and L.P. Carloni, "On the Design of a Photonic Network-on-Chip," *First International Symposium on Networks-on-Chip (NOCS'07)*, 2007, pp. 53-64.

[11] J. Balfour and W.J. Dally, "Design tradeoffs for tiled CMP on-chip networks," *International Conference on Supercomputing*, Cairns, Queensland, Australia: 2006, pp. 187 - 198.