DLB: Dynamic Lane Borrowing for Improving Bandwidth and Performance in Hybrid Memory Cube

Xianwei Zhang*, Youtao Zhang* and Jun Yang†

*Computer Science Department, University of Pittsburgh, Pittsburgh, PA, USA
{xianeizhang, zhangyt}@cs.pitt.edu
†Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA, USA
juy9@pitt.edu

***Abstract*—The Hybrid Memory Cube (HMC) is an innovative DRAM architecture that adopts 3D-stacking to improve bandwidth and save energy. An HMC module adopts separate receive and transmit lanes and thus may achieve the maximal memory bandwidth only if data can be driven at full speed in both directions. However, due to the natural read and write imbalance in modern applications, the effective memory bandwidth utilization is often low, leading to suboptimal system performance.**

In this paper, we propose DLB (dynamic lane borrowing) that dynamically tracks link utilization and partitions the lanes in one link between receive and transmit directions. DLB allocates more lanes to transmit if servicing read-intensive applications. With more lanes allocated to either direction, DLB reduces the lane contention along that direction and thus the average memory access latency. Our experimental results show that DLB improves the bandwidth utilization by 10.4% on average, reduces the average utilization gap in two directions from 35.6% to 12.8%, and saves execution time by as much as 22.3%.

***Keywords*-Hybrid memory cube, Serial link**

I. Introduction

Modern applications exhibit large demands for high memory bandwidth, i.e., more memory channels and larger bandwidth per channel [1], [2]. The parallel link design in conventional DDRx has many pins per channel, which restricts the number of channels and the total bandwidth. To alleviate this problem, recent memory architectures, such as Fully-Buffer DIMM (FB-DIMM) [3], Buffer-on-Board (BOB) [4] and Hybrid Memory Cube (HMC) [5], [6], [8], adopt high speed serial links to provide large memory bandwidth.

HMC is a memory architecture that leverages 3D stacking for latency, power, and energy reduction. One HMC module supports up to four high-speed serial links while each link consists of 16 receive (Rx) lanes and 16 transmit (Tx) lanes. Each lane can provide 12.5-30Gb/s data transfer speed, resulting in 200-480GB/s aggregated bandwidth [8]. As shown in Figure 1. The maximum HMC bandwidth is over 20 times the memory bandwidth of DDR4 and exceeds the bandwidth of high-end GPUs [7].

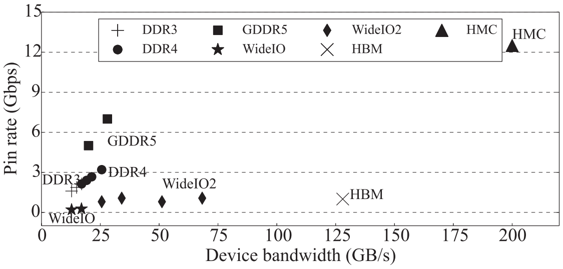


Fig. 1: Pin rate and device bandwidth comparison of different DRAM-based technologies (data from JEDEC [7].

The HMC link design adopts uni-directional lanes, i.e., a lane is dedicated to either transmit (Rx) or receive (Tx) data. This is different from the bi-directional DDRx data bus whose direction can be altered at runtime by the memory controller, after paying turn-around penalty. For an HMC link, its maximal HMC bandwidth can only be achieved if data is driven at full speed in both directions. However, modern applications have natural imbalance on the number of read and write requests. Figure 2 shows a set of applications with different percentages of reads and writes. The experimental settings can be found in Section IV. For the cases with unbalanced read and write requests, Tx lanes can be overloaded while Rx lanes are idle, and vice versa. This leads to low memory bandwidth utilization and suboptimal system performance.

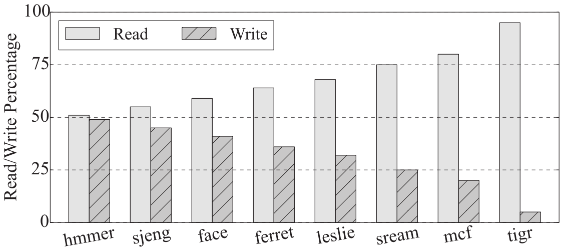


Fig. 2: Applications exhibit different Read/Write ratios.

FB-DIMM [3] and BOB [4] accommodate this imbalance by allocating more lanes for sending the data to CPU. Alternatively, lanes/links may be turned off if they are mostly idle [9], [10], which helps to reduce power consumption but not to

[1] This work was funded in part by the National Science Foundation under grants CCF-1422331 and CNS-1012070.

978-1-4673-7166-7/15/$31.00 ©2015 IEEE

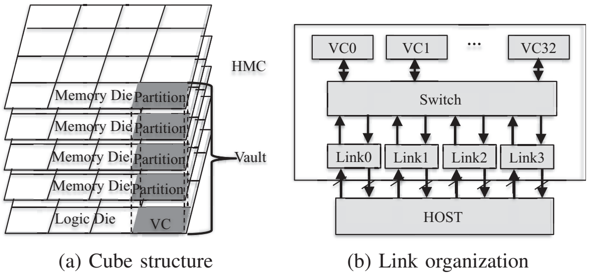
improve bandwidth utilization. To summarize, even though studies have shown that HMC is sensitive to read/write ratio [5], [11], no existing scheme targets at maximizing memory bandwidth utilization based on the skewed read/write ratio in modern applications.

In this paper, we propose DLB, a dynamic lane borrowing scheme to improve memory performance through better HMC bandwidth utilization. DLB dynamically tracks lane utilization and partitions the lanes between receive and transmit directions. An HMC link has more lanes allocated as transmit lanes if servicing read-intensive applications, and as receive lanes if servicing write-intensive applications. With more lanes allocated to either direction, DLB reduces the lane contention along that direction and improves the system performance. We evaluate two alternative designs, `WideLink` and `ExtraLink`, and compare their hardware costs and performance benefits. Our experimental results show that DLB can improve the bandwidth utilization by 10.4% and reduces the execution time by 11.3%, on average.

The rest of the paper is organized as follows: Section II introduces the HMC background. Section III elaborates the proposed designs. Section IV and V present the experimental methodology and analyze the results, respectively. We discuss the related work in Section VI and conclude the paper in Section VII.

II. Background

HMC is an innovative DRAM architecture that adopts 3D stacking to stack multiple memory dies and one logic die (at the bottom), as shown in Figure 3(a). The logic die handles controls such as DRAM sequencing, refresh, data routing and error correction; and the DRAM dies process data only [6]. Within an HMC, the memory dies are segmented vertically into vaults, and each vault has a memory controller in the logic die that manages all memory reference operations within that vault. Each vault contains partitions with several banks. Taking a 4GB HMC as an example, the package has 4 memory dies that are organized as 32 vaults while each vault has four 2-bank partitions, leading to 256 total banks.



(a) Cube structure (b) Link organization

Fig. 3: The HMC architecture [8].

HMC v2.0 supports up to four links to the controller interface, as illustrated by Figure 3(b). By default, each HMC link consists of 16 Tx lanes and 16 Rx lanes. Commands and data are transmitted over the lanes using a packet-based protocol where the packets consists of 128-bit flow units called ”FLITs” [8]. The FLITs are transmitted through the links with multiple functions and layers involved.

The data payloads to be transmitted over serial links can be 16B, 32B, 46B, or 128B long. For each payload, the **transaction and link layer** generate one packet by adding 16B overhead (i.e., 8B head and 8B tail for each packet). The packet is then chopped into FLITS, saved in the buffer, and sent in order to the **physical layer**; upon receiving the FLITS, the physical layer serializes the FLIT and drives them across the link interface in bit-serial on each lane, which is operated by the striping logic. After receiving the data, the physical layer deserializes each lane and recreates the FLITs, which are then sent to the **link layer**; the **transaction layer** stores the FLITs until the packet transmission is done, and then the FLITs are parsed, evaluated and then forwarded internally.

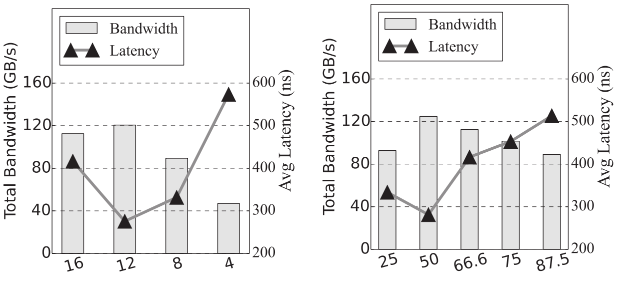
Either the requester or the responder needs to handle both Tx and Rx in each layer. Given that the components in the requester and those in the responder are fully symmetric, we only examine the responder in this paper. Since our proposed designs do not affect transaction and link layers, we focus on the enhancements to the physical layer.

III. The DLB Details

A. Motivation

The current HMC design adopts uni-directional lanes and thus idle lanes cannot be used even if the opposite direction is overloaded. To achieve higher memory bandwidth, a naive design is to fabricate more lanes in both Rx and Tx directions. This is not preferred as pins are precious resource whose count is difficult to increase and whose area budget is difficult to reduce [4]. In this paper, we exploit imbalanced lane utilization to partition the lanes for better performance.

Figure 4 studies the relationship between read/write ratio and memory bandwidth utilization. We used synthetic traffic to stress the system (the setting details are in Section IV).



(a) Impact of request lanes (b) Impact of read portion

Fig. 4: The relationship between read/write ratio and achievable memory bandwidth: (a) we vary lane partition with fixed 2/3 read requests; (b) we vary the percentage of read requests with even lane partition.

(1) In Figure 4(a), 2/3 of the requents sent to the HMC are reads while the rest are write requests. We vary the link configuration by allocating different numbers of lanes for Rx (x-axis). The rest are used for Tx. From the figure, we

found that the even lane partition (i.e., Rx gets 16 lanes) achieves sub-optimal result; the partition that allocates 12 Rx lanes and 20 Tx lanes (i.e., borrowing 4 Rx lanes) achieves the best bandwidth; and borrowing more than four Rx lanes degrades the performance due to increased contention on Rx lanes.

(2) In Figure 4(b), we evenly partition the Rx and Tx lanes, i.e., each has 16 lanes. We then vary the percentage of read requests from 25% to 87.5%, which gradually increases the load on Tx lanes. From the figure, the achievable bandwidth peaks at 50% read percentage [2] , i.e., there are equal number of read and write requests.

The study illustrates that, to achieve higher bandwidth and better performance, lanes should be partitioned according to the read/write ratio at runtime.

B. *DLB: Dynamic Lane Borrowing*

1) Lane Grouping Strategy: We next elaborate DLB, the dynamic lane borrowing scheme to maximize memory bandwidth utilization. Given that the even lane partition in the baseline HMC has supporting circuits for 16 Rx lanes and 16 Tx lanes, DLB needs to add additional hardware to enable data transmission through more lanes in either direction. Depending on the hardware overheads and the impact on timing and signal integrity, we next evaluate two alternative design choices. In the discussion, we focus on the responder side to simplify the discussion, and only borrow Rx lanes for Tx use. The scheme itself adopts symmetric link design and allows lane borrowing in either direction.

Constructing a wide single link. The first design alternative is to combine borrowed lanes with existing lanes to form one wide Tx link, e.g., borrowing 4 Rx lanes results in a 12-lane Rx and 20-lane Tx link. Figure 5 illustrates the hardware enhancement that connects borrowed lanes to the existing striping logic. Given a wider Tx link, a 128-bit FLIT can be transmitted using smaller number of UIs [8] (Unit Interval, which is the time to transmit one bit over a single lane, or m bits over a m-lane link).

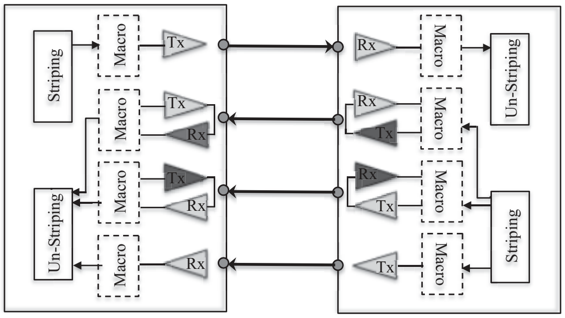


Fig. 5: Constructing a wide Tx link from borrowed lanes.

A major drawback of adopting wide link is *lane-to-lane signal skew*, i.e., when transmitting m bits simultaneously using m lanes, these bits may not be received at exactly the same time. The difference is referred to as lane-to-lane skew [12], [13]. The skew latency for 16-lane HMC link is 1066ps for receivers [8] at the receiver side, which contributes directly to the memory latency. As the skew latency depends mainly on the number of lanes (denoted as L) [13], we approximate the skew as $L/2$ UIs in this paper, which satisfies the constraints defined in [8]. Therefore, the design tradeoff of a wide link design is between increased skew latency and smaller number of UIs per FLIT transmission.

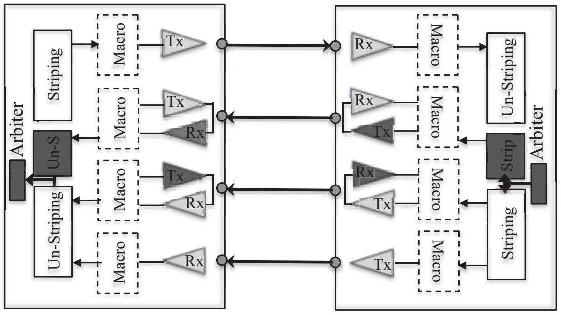


Fig. 6: Constructing an extra narrow Tx link.

Constructing an extra narrow link. The second design alternative is to construct an extra Tx link using the borrowed lanes. As shown in Figure 6, extra striping and un-striping logics are added to enable the extra link. Given the extra link is a narrow link, the lane-to-lane skew can be conservatively set as the baseline. Given the two Tx-link design, DLB enhances the link layer by adding simple arbiters such that packets may be split to FLITs for either link. Since the arbiters directly forward the received FLITs without buffering, its overhead is negligible [14].

In this section, we discuss the hardware enhancement and its overhead.

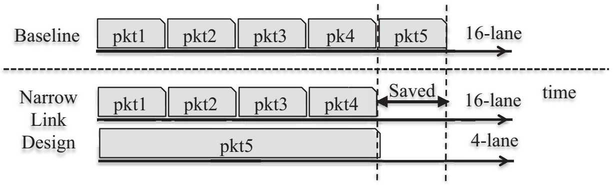


Fig. 7: Improve bandwidth utilization by using two links.

Figure 7 compares the transmission of narrow link design against baseline. In the example, we assume that DLB borrows four lanes from Rx to Tx; DLB has five packets to send; and each packet is split into 5 FLITS. 4 of 5 packets are sent through the existing Tx link while the other is sent using the extra link. From the figure, the baseline needs 5×5×128b/16 = 200UIs while the extra link design needs 160UIs only, showing 25% improvement. That is, the narrow link design not only shortens the transmission latency but also alleviates the contention along the response path, resulting in better bandwidth utilization and improved performance.

2) Hardware Overheads:

[2] Note, the strict maximum is achieved at some value slightly lower than 50% as request links transfer read/write requests as well as write data.

Area Overhead. To enable DLB, we add additional lane supporting logics to support borrowed lanes operating in opposite directions; and add extra FLIT striping/un-striping logic. DLB only affects the physical layer but not the transaction and link layers, and Tx/Rx logics are directly added to the existing lane macros and the extra small-size striping/unstriping is shared by all lanes. The area for one Rx/Tx lane is 0.155-0.24mm^2 [17], [16], [15] such that a 16-lane link occupies less than 10mm^2. We estimated additional area overhead is about 15% of the baseline. Given that a CPU or HMC chip often has several hundreds of mm^2 [5], DLB introduces modest area overhead.

Power Overhead. According to [15], Rx lane dissipates 5.1mW while Rx dissipates 6.6mW. They are negligible compared to 11W power consumption of HMC [5]. Since we did not add extra physical lanes, the power demand for pins stays the same.

Signal Integrity. One important design factor for high speed serial link is its signal integrity (SI), i.e., if the signal is reliable at high rate [18]. Recent studies [19] analyzed the lane SI for serial links. Likewise, PCIe bus can group up to 32 serial link lanes for providing multiple bandwidth — JESD204 [13]. Since DLB changes the direction but not the type of borrowed lanes, the quality of SI stays high [20].

C. Dynamic Management

We next discuss the lane management policy at runtime. There are two design choices: a semi-dynamic scheme may statically profile each workload, find its best lane partition, and reconfigure Rx/Tx lanes before program execution; a dynamic scheme tracks link and lane utilization at runtime, and dynamically reconfigures Rx/Tx lanes for achieving better system performance .

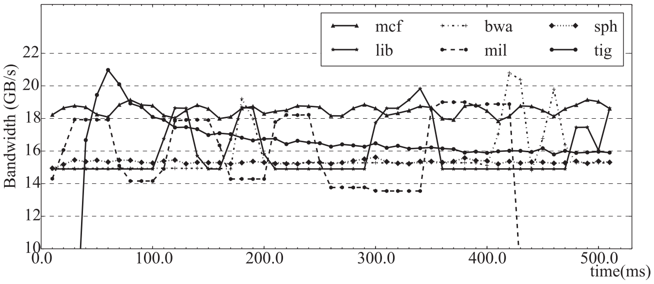


Fig. 8: Bandwidth change at runtime.

Figure 8 studies the dynamic memory bandwidth demands of different benchmark programs. From the figure, some programs, e.g., mcf, stay stable after the warm up phase, while others, e.g., lib, periodically demand large memory bandwidth. The figure also confirms that there does not exist one lane partition fitting all types of programs. To maximize memory bandwidth, it is preferable to devise a dynamic scheme that reconfigures Rx/Tx lane partition during the program execution.

```
while true do
    wait for $t_{epoch}$;
    sample bandwidth utilization on Tx/Rx links as $U_{prev\_Tx}$
    and $U_{prev\_Rx}$;
    calculate utilization gap between Tx and Rx links,
    $U_{gap} = |U_{prev\_Tx} - U_{prev\_Rx}|$;
    if (5 back_and_forth lane changes were detected) then
        pause for 10 epochs;
    end
    if ($U_{prev\_Tx} > U_{hw}$ and $U_{gap} > U_w$) then
        calculate needed lanes on Rx link as
        $U_{prev\_Rx} \times \#Rx\_lanes + guard\_band$;
        estimate the borrowing lanes to Tx link with the
        maximum limit of 8, and then floor to 2/4/8;
        update the Tx/Rx link configurations;
        return
    end
    if ($U_{prev\_Rx} > U_{hw}$) and $U_{gap} > U_w$) then
        calculate needed lanes on Tx link as
        $U_{prev\_Tx} \times \#Tx\_lanes + guard\_band$;
        estimate the borrowing lanes to Rx link with the
        maximum limit of 8, and then floor to 2/4/8;
        update the Tx/Rx link configurations;
        return
    end
end
```

Algorithm 1: Utilization based Lane Partition

The dynamic partition management algorithm is presented in Algorithm 1. The algorithm tracks link/lane utilization within each time epoch, and triggers reconfiguration at the end of each epoch if the imbalance between Rx and Tx reaches a threshold. It then calculates the preferred number of lanes and marks the rest as those that can be borrowed to service the other direction.

When determining the number of available lanes, DLB saves two extra lanes than necessary to the current direction. This is referred to as *guard band*. By preventing borrowing too many lanes to the other direction, DLB helps to gradually alleviate the contention in one direction without stressing the contention in the other direction.

Given that lending lanes increases contention and thus the likeliness of borrowing lanes, a naive dynamic design may introduce **thrashing**, i.e., the direction of a subset of lanes may be switched back and forth in consecutive epochs. In our design, we prevent thrashing by pausing DLB for ten epochs if lanes changes direction differently in five consecutive epochs.

In this paper, we set epoch to be 10ms long, specify the high watermark U_{hw} of the overloaded link as 80%, and the gap watermark U_w as 20%. We study the performance impact when varying the values of these parameters.

1) Runtime Overheads: The runtime overheads of DLB come from the dynamic link reconfiguration and the lane utilization profiling.

For dynamic link reconfiguration, recent prototypes showed that the direction of high speed serial links can be turned around in as fast as 10ns [21]. In this paper, we made a more conservative assumption — the latency of link reconfiguration is 100ns. Transmission on both directions are paused during this period, and resumed with new link settings afterward. Given that one reconfiguration epoch is 10ms, the link reconfiguration latency is negligible.

For lane utilization profiling, similar to [11], DLB needs to calculate the percentage of cycles that the lanes are used for data transmission in either direction. In this paper, we consider both the number of transmitted FLITS and the number of lanes in each epoch. The more FLITs transmitted, the better link utilization. On the other hand, since the length of a FLIT is a multiple of 8-bit, using a 18-lane Tx link often is as fast as using a 16-lane link, which results in low link utilization.

For HMC links with default 12.5Gb/s bandwidth, transmitting a 16B FLIT needs 1.28ns link time. Within one 10ms epoch, a link can transmit up to 7.82 millions FLITs in one direction. We record the number of transmitted FLITs in four 32-bit counters for Rx, Tx, and the additional sub-links in both directions (if using narrow link design).

IV. Experimental Methodology

We evaluated the proposed designs and compared them with the baseline implementation using an in-house HMC simulator. It was developed based on two existing popular memory simulators — USIMM [22] and BOBSim [4]. USIMM is a detailed DRAM simulator with a ROB front end, and BOBSim encapsulates all aspects of the ”buffer-on-board” (BOB) memory system, which works very similarly to HMC. Our simulator is a detailed cycle-accurate one modeling HMC behaviors.

System Configuration The parameters of system configurations are shown in Table I. We modeled an 8-core [9] processor running at 2.5 GHz. The memory system is configured as a 4GB HMC cube with 4 DRAM dies and 32 vaults [8]. For the link, we used one for experiment purpose by manually disabling others as [10]. The one-link test is a reasonable setting for HMC study because: (1) nowadays applications can hardly take full use of available bandwidth, and our workloads here is close to the bandwidth of one single link; (2) multiple HMC devices can be chained together as a cube network and each link is configured as either a host link or a pass-through link, leading to only one or two links are connected to the processor [8], [23], which is similar to the case here.

TABLE I: System Configuration

Processor	8 cores, 2.5GHz, 10 stage pipeline, 128-entry ROB, fetch/retire width=4/4
Last Level Cache	private, 512KB/core, 4MB in total [22] cache line size: 64B
Link	1 full-duplex link, 12.5 Gb/s lane speed default width: 16 Tx and 16 Rx lanes
Vault Controller	32-entry command queue, 1024B read data queue scheduling: FRFCFS, closed-page[11], [6]
HMC	4GB, 32 vaults, 4 2-bank partitions/vault page size: 256B [11], [6] TSV: 32 TSVs/partition [6], 2.5Gb/s packet: 64B data payload, 16B overhead timings: tCK=0.8ns, tRP=18, tRCD=18, tCL=18, tRAS=35, tWR=19, tCCD=7
Lane Management	epoch:10ms, turnaround time: 100ns high watermark of link utilization: 80% utilization gap: 20%, guard band: 2 number of allowed borrowing lanes: 2/4/8

For the controller part, we assume the vault throughput of 10GB/s [8] is evenly divided among the 32 TSV data lanes, with each as 2.5Gb/s. If the transmission is double rate, then TSV frequency is 1.25 GHz ($t_{CK} = 0.8ns$). Here, we get the equivalent timing constraints, partially shown in Table I, based on the values used by [11], [23], where $t_{CK} = 1.25ns$.

Workloads We used multiple benchmarks from SPEC CPU 2006, Biobench [24] and STREAM [25]. The memory characteristics are summarized in Table II, and the workloads used for simulation are as shown in Table III. The traces are collected after skipping warmup phase. Each core executes 20M memory read and write requests, which translates into several billions processor cycles and instructions for the 8-core scenario. We executed these benchmarks in rate mode, and computed the execution time as the time to finish all benchmarks of a workload.

TABLE II: Benchmark Characteristics

Suite	Benchmark	Read MPKI	Write MPKI
SPEC	mcf	69.46	16.83
	libquantum	25.90	3.91
	bwaves	18.71	1.08
	milc	3.81	1.32
	sphinx	13.23	0.84
BIOBENCH	mummer	26.43	5.72
	tigr	33.87	4.34
MICRO-BENCH	stream	35.07	11.69

TABLE III: Workloads

Description
mcf: mcf×8, lib: libq×8, bwa: bwaves×8, sph: sphinx×8, mum: mummer×8, tig: tigr×8, str: stream×8
mix1: mcf×2, libq×2, mummer×2, tigr×2 mix2: bwaves×2, milc×2, sphinx×2, stream×2 mix3: mcf×2, libquantum×2, sphinx×2, stream×2 mix4: bwaves×2, milc×2, mummer×2, tigr×2

V. Results

In the experiments, we evaluated the following schemes:

— `Baseline`. This is the baseline scheme. Each link is composed of 16 Rx lanes and 16 Tx lanes.

— `WideLink`. This scheme enables Rx lanes to be borrowed to Tx links. The borrowed lanes are used to widen the Tx link.

— `ExtraLink`. This scheme groups borrowed lanes into an extra narrow Tx link.

Next, we compare the schemes on execution time, memory access latency, bandwidth utilization, and study their sensitivity to different configurations.

A. Impact on Execution Time

Figure 9 compares the execution time with different schemes. All the results are normalized to `Baseline` and the geometric mean (`Gmean`) of all evaluated workloads is presented as well. Compared to `Baseline`, lane borrowing schemes `WideLink` and `ExtraLink` widen the response path, speedup read accesses, and thus shorten the program execution. The figure shows that `WideLink` and `ExtraLink`

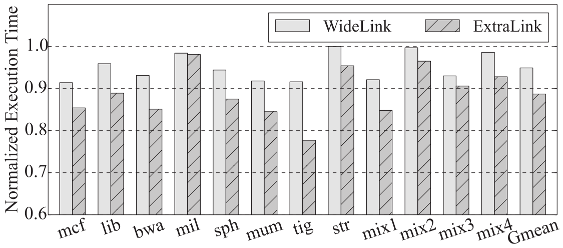


Fig. 9: Comparing the execution time.

achieve better performance for almost all workloads. On average, WideLink reduces the execution time by 5.1% while ExtraLink achieves a much larger 11.3% reduction. For memory intensive workload with more read requests, e.g., mcf, mum and tig, the improvement is more significant — ExtraLink achieves 22.3% improvement for tig. For non-memory intensive workloads or workloads with balanced read/write requests, such as $milc$ and str, the performance gain is lower, e.g., only 2% for $milc$.

The giant gap between WideLink and ExtraLink is attributed to two reasons: (1) WideLink introduces larger skew delay which contributes directly to memory latency; (2) as the FLIT size is kept at 128 bit, which may lead to inefficient lane use in WideLink, e.g., the elapsed time to transfer a FLIT is the same 8UIs using either 16 or 18 lanes. Nevertheless, WideLink has the advantage of less added logics and thus smaller overhead than ExtraLink.

B. Impact on Latency

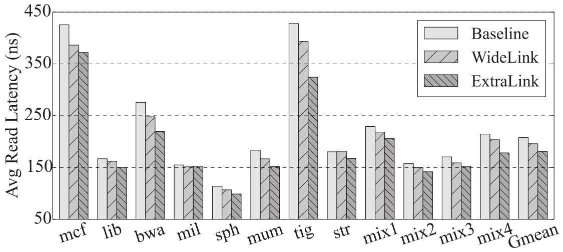


Fig. 10: The read latency comparison.

Figure 10 shows the average memory read latency for different schemes. The read latency consists of HMC bank access time and queuing time. While bank access time is stable, queuing time strongly depends on the link contention. From the figure, on average, WideLink brings the latency from 208ns in Baseline down to 195ns, which is a 7% reduction. As expected, ExtraLink shows a greater decrease of 14%.

C. Impact on Bandwidth

The performance improvement is achieved through better bandwidth utilization. In Baseline, the request link and response link have equal theoretical bandwidth of 25GB/s (i.e. 12.5Gb/s/lane×16 lanes); in WideLink and ExtraLink, while the total bandwidth is still 50GB/s, the division is dynamically adjusted through lane borrowing reconfigurations.

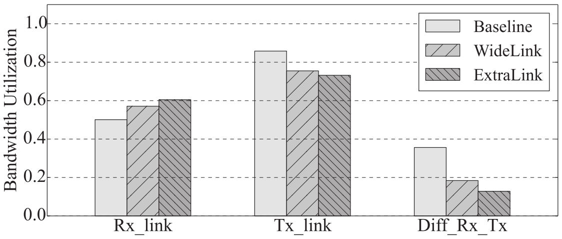


Fig. 11: Comparison of bandwidth utilization.

In the experiments, we found that, in Baseline, while response (Tx) bandwidth achieves a high utilization of over 80% on average, request (Rx) bandwidth is less than 50% for many workloads, which indicates a severe waste. The proposed design efficiently migrates the excessive Rx bandwidth to the Tx link by improving the utilization in Rx direction, and mitigating the contention on Tx side. Figure 11 shows the average bandwidth utilization, which is calculated by averaging the utilization of all epochs during execution, of Rx link, Tx link and the difference for all three schemes. Overall, as shown in Figure 11, ExtraLink improves the utilization by 10.4%, and the difference between Rx and Tx link utilizations is reduced from 35.6% in Baseline to 18.4% in WideLink, and further to 12.8% in ExtraLink.

D. Runtime Reconfigurations

We then studied lane borrowing during the entire execution, as shown in Figure 12. We reported the number of lanes used as Rx lanes while the rest are used as Tx lanes. While Baseline keeps the default lane setting, WideLink and ExtraLink schemes dynamically adjust the lane configurations from the initial even lane partition.

From the figure, the number of Rx lanes is relatively stable because: (1) the workload phases have relatively stable memory access patterns and the chosen 10ms period adapts well to phase changes; (2) the conservative borrowing strategy (e.g., maximum 8 borrowing lanes and a 2-lane guard band) prevents over-reaction and thus helps to avoid thrashing.

E. Responsiveness Testing

It is well known that most applications have different phases in the execution, and the exhibited memory access behaviors are relatively stable within each phase. To further test the responsiveness of our designs, we created a synthetic workload using the representative ones covered in Figure 13, and executed them in round robin fashion to expose changing loads to the links. The selected workloads are lib, mcf, sph and str, with each being executed for a period of 30ms (i.e., three times of reconfiguration epoch) in a round.

Figure 13 shows the dynamic lane change in the 1200ms execution. As expected, ExtraLink experiences frequent lane reconfigurations. Nevertheless, the inefficiency on wide link makes WideLink stay at a relative stable state. Because of the frequent changing workload patterns, the improvements are diminishing to 8.4% for ExtraLink and 2% for WideLink.

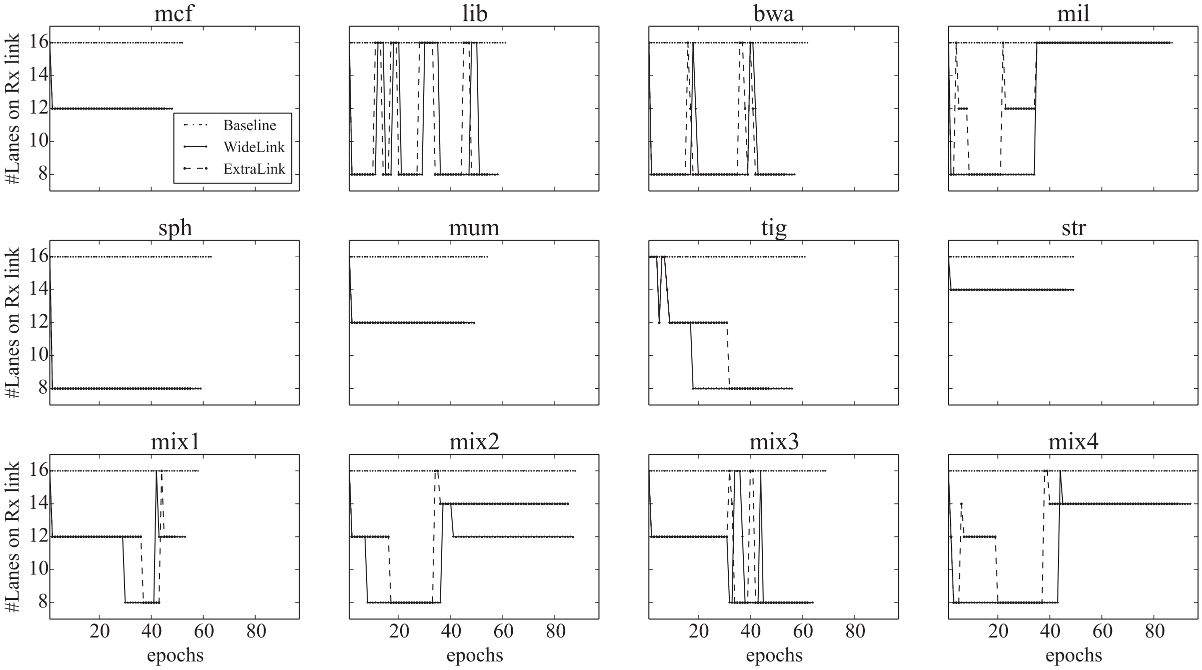


Fig. 12: Rx lanes reconfiguration at runtime (X-axis is set to be 100 epochs while some executions are shorter).

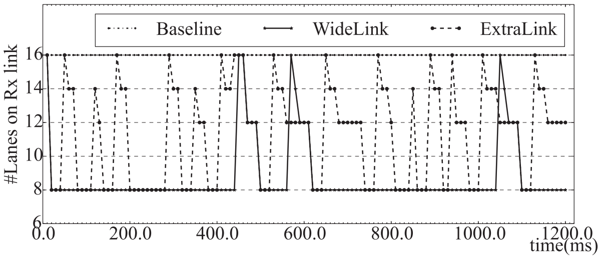


Fig. 13: Responsiveness testing with synthetic workload.

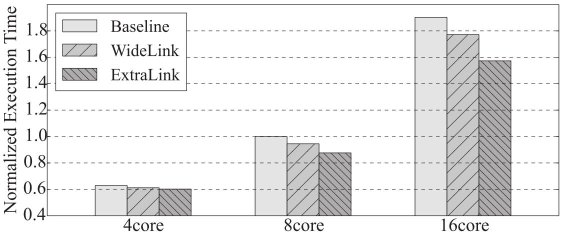


Fig. 14: Sensitivity on the number of cores (showing the execution time of one single core).

F. Sensitivity Studies

System performance largely depends on the request intensiveness, which is strongly related to the number of cores and packet size. To study the performance sensitivity on these two factors, we varied the number of cores from 4 to 16, and transaction size from 16B to 128B. In addition, the dynamic policy settings were also considered. All the performance values are normalized to the baseline of 8 cores, 64B transaction size, and default 16/16 lane setting.

1) Varying the Number of Cores: HMC links are expected to transfer more requests and data with increasing number of cores, which increases link contention and prolongs execution time. As shown in Figure 14, compare to the 8-core baseline, a 4-core system has better execution time for the execution on each single core. This is because of shorter queuing delay when the HMC module receives few requests.

The benefits of lane borrowing goes up with increasing number of cores. For example, `ExtraLink` achieves 4.8% improvement on a 4-core system but 18.4% improvement on a 16-core system.

2) Varying Packet Size: HMC supports data payload of size from 16B to 128B, with extra 16B overhead to compose the packet. The data size determines the link occupation time. We swept the data size from 16B to 128B to study its impact on system performance. As Figure 15 shows, the execution time is lengthened as the data size becomes larger. We observed better performance improvement when using larger data size. For the setting of 128B, `WideLink` reduces the execution time by 8.6%, while `ExtraLink` reduces 18.6%.

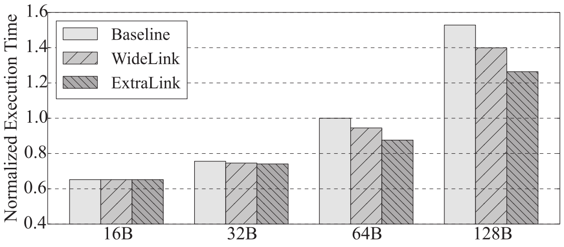


Fig. 15: Sensitivity on data payload size.

3) Varying Dynamic Epoch and Watermarks: We next studied the impact of epoch size. Figure 16 compares the performance of different schemes with epoch sizes ranging from 1ms to 100ms. When increasing the epoch size from 1ms to 15ms, we observed 2% performance difference, which indicates that performance is insensitive to epoch size. However, when the epoch is further increased to 100ms, compared to 10ms setting, we observed performance reduction of 2.7% and 4% for `WideLink` and `ExtraLink`, respectively, indicating that the 100ms-epoch responds too slow to the change of memory access patterns.

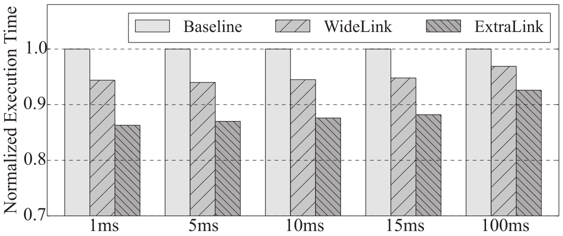


Fig. 16: Sensitivity on dynamic reconfiguration epoch.

In addition, we also performed sensitivity studies on high watermark of bandwidth utilization U_{hw}, and utilization gap watermark U_w, and found moderate impacts on performance.

VI. Related Work

As HMC is an emerging DRAM architecture, limited work has been done to study its benefits and issues on performance, bandwidth or power. Khurshid *et al.* [26] proposed thermal mitigation using data compression. Han *et al.* [27] exploits data-pattern dependent cell retention to reduce refresh operations in HMC. Lu *et al.* [28] designed an efficient packet-based memory system to solve the high packet overhead. Pugsley *et al.* [10] presented a high-level Near Data Computing (NDC) architecture using HMC devices. To better manage the link power consumption, Ahn *et al.* [9] proposed a scheme to dynamically put a subset of off-chip links into sleep mode. Kim *et al.* [23] proposed an innovative distributed memory-centric network architecture to efficiently utilize the processor and HMC bandwidth. The network was later extended to multi-GPU system [29].

Recent studies showed that HMC is sensitive to the read/write ratio [5], [11]. However, to the best our knowledge, no scheme has been proposed to improve the bandwidth utilization based on memory access patterns. DLB works similarly to reconfigurable switching path [30] in Network-on-Chip (NoC), and the recently proposed DDRx pin-switching scheme [14]. While the former is specially designed for on-chip communication, the latter does not adapt read/write imbalance of real applications.

VII. Conclusion

In this paper, we proposed DLB, a dynamic lane borrowing scheme to improve bandwidth utilization and system performance in HMC architecture. We studied two alternative designs with different design costs and performance benefits, and enabled dynamic partition by tracking link utilization at runtime. Our experimental results showed that the proposed scheme improves bandwidth utilization of HMC links by about 10.4%, with a performance improvement up to 22.3% (11.3% on average).

References

[1] S. W. Kechler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," in *IEEE Micro*, 2011.

[2] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross, "Hardware partitioning for big data analytics," *IEEE Micro*, 2014.

[3] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, "Fully-buffered DIMM memory architectures: understanding mechanisms, overheads and scaling," in *HPCA*, 2007.

[4] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer-on-board memory system," in *ISCA*, 2012.

[5] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hot Chips*, 2011.

[6] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *VLSIT*, 2012.

[7] "JEDEC specifications (DDR3, DDR4, GDDR5, Wide I/O, HBM, HMC)" in *http://www.jedec.org/*.

[8] "Hybrid memory cube specification 2.0," http://www.hybridmemorycube.org/.

[9] J. Ahn, S. Yoo, and K. Choi, "Dynamic power mangement of off-chip links for hybrid memory cubes," in *DAC*, 2014.

[10] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu *et al.*, "NDC: analyzing the impact of 3D-stacked memory+logic devices on mapreduce workloads," in *ISPASS*, 2014.

[11] P. Rosenfeld, "Performance exploration of the hybrid memory cube," Ph.D. dissertation, University of Mayland, College Park, 2014.

[12] R. Budruk, D. Anderson, and T. Shanley, *PCI express system architecture*. Addison-Wesley Professional, 2003.

[13] "JEDEC standard - serial interface for data converters," www.jedec.org/sites/default/files/docs/JESD204B.pdf.

[14] S. Chen, Y. Hu, Y. Zhang, L. Peng, J. Ardonne, S. Irving *et al.*, "Increasing off-chip bandwidth in multi-core processors with switchable pins," in *ISCA*, 2014.

[15] K. Fukuda, H. Yamashita, G. Ono, R. Nemoto, E. Suzuki, T. Takemoto *et al.*, "A 12.3mW 12.5Gb/s complete transceiver in 65nm CMOS," in *ISSCC*, 2010.

[16] F. Spagna, L. Chen, M. Deshpande, Y. Fan, D. Gambetta, S. Gowder *et al.*, "A 78mW 11.8Gb/s serial link transceiver with adaptive RX equalization and baud-rate CDR in 32nm CMOS," in *ISSCC*, 2010.

[17] J. F. Bulzacchelli, T. O. Dickson, Z. T. Deniz, H. A. Ainspan, B. D. Parker, M. P. Beakes *et al.*, "A 78mw 11.1Gb/s 5-tap DFE receiver with digitally calibrated current-integrating summers in 65nm CMOS," in *ISSCC*, 2009.

[18] J. N. Tripathi, R. K. Nagpal, and R. Malik, "Robust optimization of serial link system for signal integrity and power integrity," in *NESEA*, 2010.

[19] M. P. Li, *Jitter, noise and signal integrity at high-speed*. Pearson Education, 2007.

[20] S. Hall and H. Heck, *Advanced signal integrity for high-speed digital designs*. Wiley-IEEE Press, 2009.

[21] V. Balan, O. Oluwole, G. Kodani, C. Zhong, S. Maheswari, R. Dadi *et al.*, "A 130mW 20Gb/s half-duplex serial link in 28nm CMOS."

[22] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee *et al.*, "USIMM: the utah simulated memory module," University of Utah, Tech. Rep., 2012.

[23] G. Kim, J. Kim, J. H. Ahn, and J. Kim, "Memory-centric system interconnect design with hybrid memory cube," in *PACT*, 2013.

[24] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng *et al.*, "BioBench: A benchmark suite of bioinformatics applications," in *ISPASS*, 2005.

[25] J. D. McCalpin, "STREAM: sustainable memory bandwidth in high performance computers," http://www.cs.virginia.edu/stream/.

[26] M. J. Khurshid and M. Lipasti, "Data compression for thermal mitigation in the hybrid memory cube," in *ICCD*, 13.

[27] Y. Han, Y. Wang, H. Li, and X. Li, "Data-aware DRAM refresh to squeeze the margin of retention time in hybrid memory cube," in *ICCAD*, 2014.

[28] T. Lu, L. Chen, and M. Chen, "Achieving efficient packet-based memory system by exploiting correlation of memory requests," in *DATE*, 2014.

[29] G. Kim, M. Lee, J. Jeong, and J. Kim, "Multi-GPU system design with memory networks," in *MICRO*, 2014.

[30] B. Ahmad, A. .T. Erdogan, and S. Khawam, "Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable MPSoC" in *AHS*, 2006.