Memory-centric System Interconnect Design with Hybrid Memory Cubes

Gwangsun Kim, John Kim
KAIST
{gkim, jjk12}@cs.kaist.ac.kr

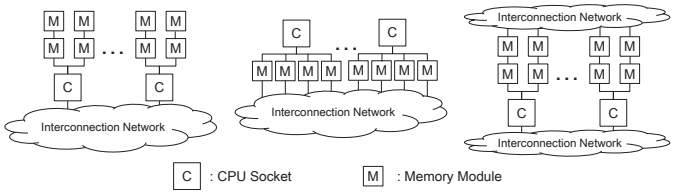
Jung Ho Ahn, Jaeha Kim
Seoul National University
{gajh, jaeha}@snu.ac.kr

***Abstract*—Memory bandwidth has been one of the most critical system performance bottlenecks. As a result, the HMC (Hybrid Memory Cube) has recently been proposed to improve DRAM bandwidth as well as energy efficiency. In this paper, we explore different system interconnect designs with HMCs. We show that *processor-centric* network architectures cannot fully utilize processor bandwidth across different traffic patterns. Thus, we propose a *memory-centric* network in which all processor channels are connected to HMCs and not to any other processors as all communication between processors goes through intermediate HMCs. Since there are multiple HMCs per processor, we propose a *distributor-based* network to reduce the network diameter and achieve lower latency while properly distributing the bandwidth across different routers and providing path diversity. Memory-centric networks lead to some challenges including higher processor-to-processor latency and the need to properly exploit the path diversity. We propose a pass-through microarchitecture, which, in combination with the proper *intra*-HMC organization, reduces the zero-load latency while exploiting adaptive (and non-minimal) routing to load-balance across different channels. Our results show that memory-centric networks can efficiently utilize processor bandwidth for different traffic patterns and achieve higher performance by providing higher memory bandwidth and lower latency.**

***Keywords*—Hybrid Memory Cube, Memory network, Memory bandwidth wall, Processor interconnect**

I. Introduction

Memory bandwidth has historically been one of the most critical system performance bottlenecks as processor performance has grown at a faster rate than has memory bandwidth growth [1], [2]. The memory bandwidth wall problem was created by both the processor chip I/O bandwidth and the DRAM channel bandwidth. Recently, the HMC (Hybrid Memory Cube) [3], [4], [5] has been proposed to improve the DRAM bandwidth and energy efficiency by 3D-stacking DRAM dies on top of a logic die that interfaces with processors or other HMCs. The aggregate bandwidth provided by a single HMC is substantial due to the large number of independent memory banks, high-bandwidth through-silicon vias (TSVs), and a high-speed I/O interface, that provides total I/O bandwidth of up to 320 GB/s in the prototype of the first generation HMC [3]. Although HMC can significantly improve the memory bandwidth, it remains a challenge to efficiently utilize the processor I/O bandwidth in a system that contains multiple HMCs. In this work, we explore the design space of a system interconnect that consists of processors and multiple HMCs and describe how to exploit the high bandwidth. We define the system interconnect [1] as a processor interconnect such as the Intel QuickPath Interconnect (QPI) [6] and HyperTransport [7].



(a) Processor-centric Network (b) Memory-centric Network (c) Hybrid Network

Fig. 1. Design space of system interconnect designs.

In traditional processor interconnects, such as QPI and HyperTransport, some of the processor channels are used to communicate with other processors or nodes; the remaining channels are used to communicate with local DRAM memory modules, as shown in Fig. 1(a). We refer to this type of system interconnect as a *processor-centric* network (PCN). In this network organization, a processor's local DRAM bandwidth is limited by the number of local memory channels. Dedicating some of the processor bandwidth only to memory channels is inevitable as low-level DRAM commands, address, and data have to be communicated through the channels. However, in HMCs, packet-based abstraction protocol is used for the processor-memory interconnect [3]. The packet includes routing information to reach the destination HMC; switching is supported by the HMC's logic die. The packet-based protocol provides different opportunities in processor-memory interconnects and enables topologies and connectivity that were not possible with traditional DIMM-based DRAM modules.

In this work, we propose an alternative organization – a *memory-centric* network (MCN), as shown in Fig. 1(b), in which all processor channels are connected to local HMCs and processor-to-processor communication is routed through the local HMCs. In addition to PCN and MCN, the two network types can be combined to create a *hybrid* network as shown in Fig. 1(c). We explore the three different network organizations shown in Fig. 1, which consist of a few processor sockets and many HMCs and evaluate their trade-off. Our work shows that simply replacing traditional DIMMs with HMCs and using a traditional PCN does not result in the best performance; we show how the MCN and hybrid approaches can better utilize processor I/O bandwidth and improve performance.

[1]The term system interconnect is also sometimes used to describe large-scale networks such as the networks found in supercomputers. However, we do not focus on such networks in this work.

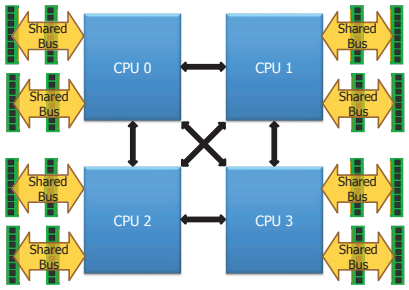


Fig. 2. A quad-socket example of the Intel QuickPath Interconnect where processors are fully connected with high-speed point-to-point links and multiple DIMMs are connected to the processor via shared buses.

In addition, we describe different network design techniques, including distributor-based networks, pass-through microarchitecture, and adaptive routing in a system interconnect design to further improve performance.

The contributions of this work include the following:

- We explore network designs in system processor interconnects with hybrid memory cubes (HMCs). Instead of a *processor*-centric network, we propose a *memory*-centric network that provides more flexible bandwidth utilization in the memory network.
- With more routers (or HMCs) than terminal nodes (or processors), we propose a *distributor*-based network that not only minimizes network diameter but also better distributes the processor bandwidth across different routers.
- With the proposed network organizations, we show how adaptive (and non-minimal) routing can enable similar throughput on both load-balanced traffic and adversarial traffic patterns.
- The zero-load latency [2] for CPU-to-CPU packets can increase with higher hop count in memory-centric networks, but we show how pass-through microarchitecture and exploiting the *intra*-HMC network port placement can minimize the zero-load latency.

II. Background

In this section, we provide background on conventional processor interconnect design, recently proposed Hybrid Memory Cubes (HMCs), and interconnection network design techniques that we leverage in this work.

A. Processor Interconnect and DRAM Interface

The Intel QuickPath Interconnect [6] and HyperTransport [7] are two popular processor interconnects used in modern computer systems. In these networks, multiple processors are interconnected via high-speed point-to-point links to provide high bandwidth. As recent processors have integrated on-chip memory controllers, DRAM devices are directly connected to the processor via a shared bus interface. Figure 2 shows an example of the Intel QuickPath Interconnect in which four processors are fully connected to each other and each processor has two memory channels. Since shared bus interconnects are used for DRAM interconnects, there is a limit to the interconnect organization for the entire system because the DRAM devices have no routing capabilities and cannot be used to create a network. However, since HMC adopts high-speed signaling as the interface and routing/switching capabilities in the logic die as we explain in Section II-B, there is a higher degree of freedom in the system interconnect design. As a result, this work explores the design space for such system interconnects with HMCs.

[2]The zero-load latency is defined as the network latency when there is no load or "zero-load" in the network.

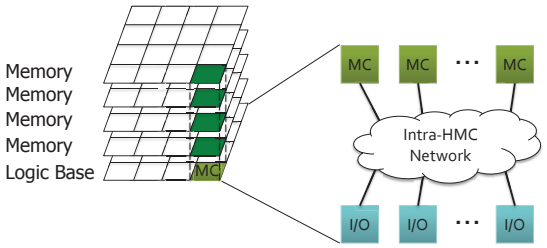


Fig. 3. Block diagram of a hybrid memory cube (HMC) that consists of several layers of DRAM dies and a logic layer on the bottom. The logic layer consists of an intra-HMC network.

B. Hybrid Memory Cube (HMC) Overview

Hybrid memory cubes (HMCs) [5] have been recently proposed to provide higher memory bandwidth while providing scalability and improving energy efficiency [3], [4]. A block diagram of an HMC structure is shown in Fig. 3; this structure consists of several DRAM layers stacked with a logic layer on the bottom. The different layers are connected with Through-Silicon Via (TSV) interconnects. Each layer of memory is partitioned into different segments and the vertical segments are grouped together to form a vertical slice or a *vault*. The logic layer on the bottom consists of the memory controllers (one for each vault), the I/Os, and a switch that interconnects the vaults and the I/Os. High-speed signaling is used to interconnect the HMCs to the CPU socket as well as to other HMCs in order to provide high bandwidth to/from the HMCs. This structure also results in an abstracted memory interface because the CPU does not necessarily need to be aware of the details within the DRAM technology. The processor communicates with an HMC-side memory controller by sending/receiving high-level memory request/response messages that include memory request types, such as read or write, memory addresses, and data, instead of sending low-level commands to the DRAM devices and being aware of the internal DRAM device timing. All detailed timing and operations regarding DRAM devices are managed by the HMC-side memory controllers on the logic die.

Within the logic layer of an HMC, a switch is needed to interconnect different vault memory controllers and the I/O ports. In the HMC system that we consider, there are 8 I/O ports and 16 vault memory controllers; thus, this switch is actually a 24×24 switch. In this work, we refer to this switch as the *intra*-HMC network, in comparison with the *inter*-HMC network that corresponds to the high-level topology used to interconnect different HMCs and CPUs. The I/O ports in the HMC can be used to connect with other HMCs or CPU sockets.

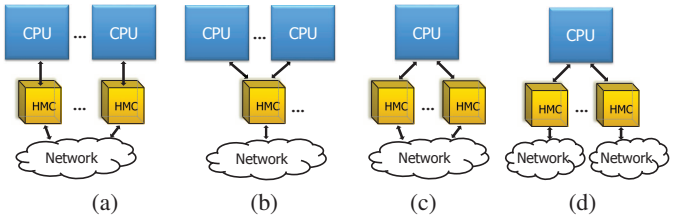


Fig. 4. (a) A baseline network design and alternative network organizations using (b) concentration, (c) distribution, and (d) channel slicing.

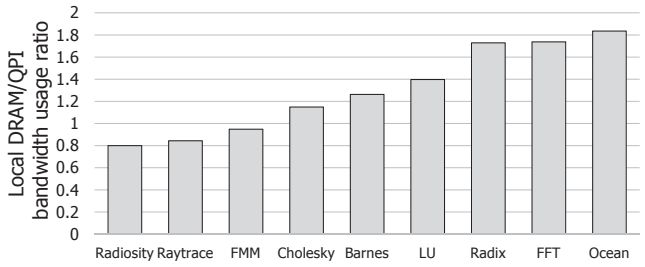


Fig. 5. The ratio of local DRAM access bandwidth vs. QPI bandwidth for SPLASH-2 [12] applications measured with a quad-socket Intel XEON E7540 machine with CentOS 6.3.

Although any HMC can be reached by a processor in the HMC networks, we assume the modern cache coherence protocol used in the DIMM-based memory systems. We assume that a processor only manages cache coherence of the address space that corresponds to its local HMCs, thus requiring sending cache coherence requests to other processor to access the processor's HMCs (or remote HMCs). We also define a *cluster* as a group consisting of a processor and its local HMCs.

C. Interconnection Network Design

With multiple HMCs in a system, an interconnection network is needed to interconnect the HMCs and the processor sockets. Concentration [8] is a technique used in interconnection networks; using this technique, network resources are shared between different components. Concentration has been used both for off-chip networks [9], [10] and for on-chip networks [11] as it reduces the number of intermediate routers and lower network cost and network latency. Examples of the networks with and without concentration are shown in Fig. 4(a,b). As opposed to Fig. 4(a), in which one router is dedicated for each processor, when using concentration, multiple processors share the same router (Fig. 4(b)).

In comparison, *distribution* is the opposite of concentration; in distribution, a node is connected to multiple routers, as shown in Fig. 4(c). By leveraging the distribution, the traffic from a single source can be distributed across different network routers. The benefits of this approach include not only better load-balancing but also lower network latency. The distributor-based networks that we leverage are different from channel-slicing [8], in which multiple channels from a node are connected to multiple parallel networks (Fig. 4(d)). A channel sliced network provides path diversity across different parallel networks but does not provide any connectivity between the parallel networks.

The baseline processor-centric memory organization used in current processor interconnects (such as those shown in Fig. 2) can be categorized as channel-sliced networks because

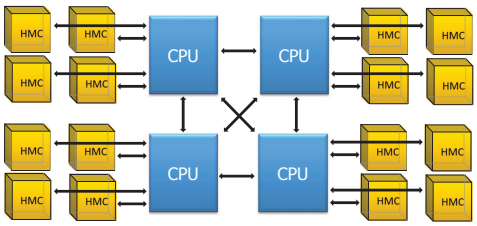


Fig. 6. Processor-centric network organization with 4 CPU sockets and 16 HMCs.

dedicated channels (or shared buses) are used for different memory DIMMs. For example, if the total CPU-memory bandwidth is B_m and the bandwidth is sliced into s channels, the channel-sliced approach only provides B_m/s bandwidth to each memory device. In comparison, the distributor-based HMC network provides bandwidth flexibility because more than B_m/s bandwidth can be leveraged for each HMC through adaptive routing.

D. Design Objectives

We identify the following design objectives for the system interconnect and consider them in designing our proposed networks.

- **Low-latency:** In general, latency has direct impact on performance and is proportional to the hop count; thus, a topology with low hop count values or network diameter can provide better system-level performance. Another unique challenge in the HMC system interconnect is that low latency is required not only between the processors but also between the processors and the HMCs.
- **High-bandwidth:** Recent multi-core trends have resulted in an increase in the number of cores being integrated in a processor. As each core requires a certain amount of memory bandwidth [13], the increase of the number of cores results in processor bandwidth demand increasing accordingly [1].
- **Flexibility:** As shown in Fig. 5, applications have different traffic characteristics, in which one application can require high processor-to-processor bandwidth while another can require high local DRAM bandwidth. It is also possible for processor cores to access the same HMC, which would result in a hotspot traffic pattern. Thus, it is important to provide high memory bandwidth and flexibility across different traffic patterns.

III. Memory-Centric System Interconnect

In this section, we explore an alternative system interconnect network design that leverages HMCs. We first describe our baseline network based on the current system interconnect design – a processor-centric network. We then propose how an alternative system interconnect organization, a *memory*-centric network, can better exploit HMCs to provide a more scalable interconnect. To further improve the performance of the memory-centric networks, we describe two techniques – non-minimal routing and pass-through implementation to increase bandwidth utilization and reduce latency.

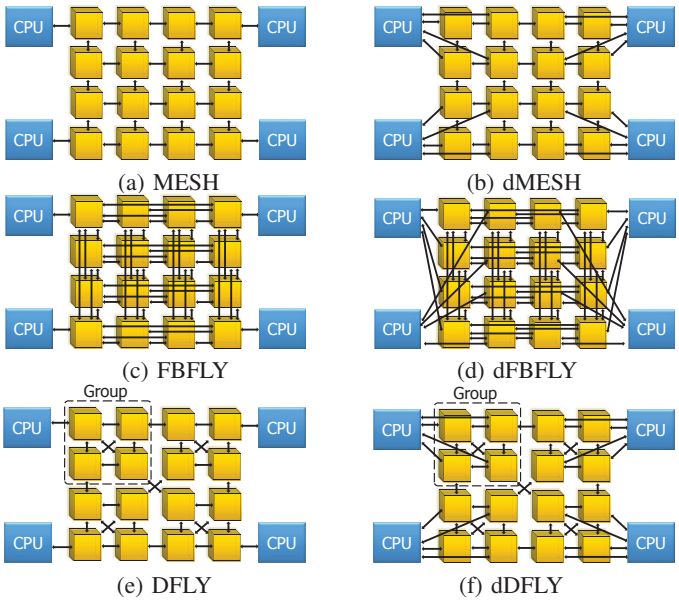


Fig. 7. Memory-centric network based on (a) mesh, (c) flattened butterfly, (e) dragonfly and (b,d,f) the same topologies with distribution.

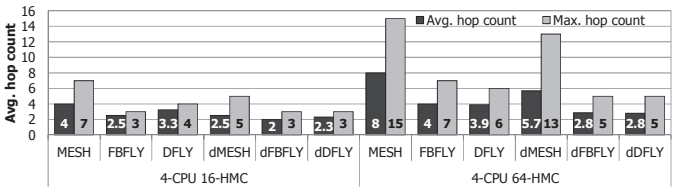


Fig. 8. Hop count comparison of different memory-centric network designs.

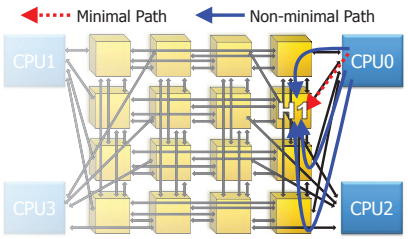


Fig. 9. Minimal and non-minimal paths between CPU0 and destination HMC (**H1**) in dFBFLY.

A. *Distributor-based Network*

Most current processor interconnects are *processor*-centric networks as each processor is directly connected to other processors. The processors are interconnected with other processors and thus, memory requests to *remote* memory are routed through other processors. An example of processor-centric networks is shown in Fig. 6 with 4 CPU sockets and 16 HMCs. Although this organization is similar to current processor interconnect (Fig. 2), it does not efficiently utilize the CPU channel bandwidth. In the example from Fig. 6, four of the seven processor channels are dedicated to HMC connection while the remaining three are used for CPU-to-CPU connection. In addition, the four channels for CPU-HMC connections in this example are dedicated channels to each HMC and cannot be flexibly utilized to provide more bandwidth to a given HMC.

To avoid this limitation, we propose a *memory*-centric network (MCN) with HMCs where the memory themselves form a network and communication among processors is routed through the memory network. Examples of different MCN organizations with 4 CPU sockets and 16 HMCs are shown in Fig. 7, which includes a mesh [3], flattened butterfly [14], and dragonfly topology [15]. By leveraging a topology such as a flattened butterfly topology or dragonfly with higher connectivity, the additional channels in the HMCs can be exploited to reduce the network hop count (Fig. 7(c,e)). However, one limitation with these networks is that the CPU is only connected to a single HMC directly. This increases the hop count as all requests need to be first routed through this single HMC and also limits the CPU-to-memory bandwidth. To overcome this limitation, we propose a distributor-based network to exploit the high CPU bandwidth while also reducing the network diameter. The distributor-based network spreads the bandwidth of the terminal nodes (i.e., CPU) across the different routers (or HMCs) as the CPU is directly connected to multiple HMCs. The examples of 4-way distributor-based networks (which we refer to as dMESH, dFBFLY, and dDFLY) are shown in Fig. 7(b,d,f). The distributor-based network is the opposite of using concentration – instead of sharing network resources, we distribute the traffic across different locations of the network. By using a distributor-based network, we can further reduce the network diameter and latency. Compared to the same flattened butterfly topology, the average hop count is reduced by 25% for uniform random traffic (and by 66.7% for an adversarial traffic pattern) as shown in Fig. 8. In addition, the full bandwidth from each socket can be fully utilized by spreading the bandwidth across the different nodes. Thus, instead of using the total bandwidth as a single, fat channel, more channels can be leveraged to create a more cost-efficient network and provide bandwidth flexibility.

Although the memory-centric network provides high bandwidth, to fully exploit the benefits of this network, non-minimal routing is needed. In addition, the CPU-to-CPU latency is increased with higher hop count in a memory-centric network. In the next two sections, we describe how adaptive routing with non-minimal routing can be used in HMC networks to exploit the bandwidth flexibility. In addition, we propose pass-through microarchitecture to minimize the latency for CPU-to-CPU communication in a memory-centric network.

B. *Increasing Bandwidth with Non-minimal Routing*

Most processor-interconnects have often used deterministic, minimal routing. For a processor-centric network organization (such as Fig. 6), there is only a single path to a particular HMC and thus, minimal routing is sufficient. [4] However, with higher connectivity in memory-centric network, non-minimal

[3]This organization is very similar to how some multicore on-chip networks are interconnected – cores are interconnected through a network and the memory controllers are attached at the corners. The main difference is that the location of the "cores" are swapped. It is also similar to how cores are interconnected to a large number of cache banks in a NUCA organization.

[4]Non-minimal routing can be beneficial in the PCN for CPU-to-CPU communication.

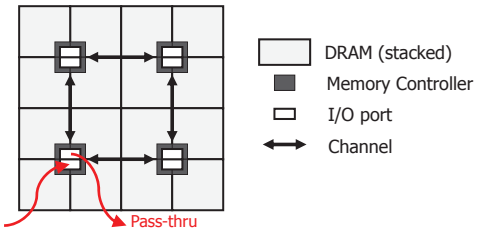


Fig. 10. Floorplan of a concentrated mesh intra-HMC network within the logic layer. The channel refers to the on-chip channel and the DRAM is located on other layers but shown for illustration purpose. A pair of I/O ports are placed nearby to minimize on-chip latency for pass-through.

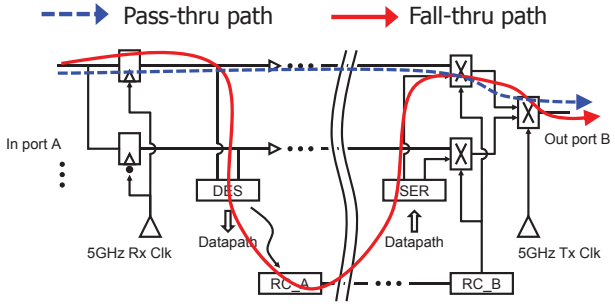


Fig. 11. Router I/O path with fall-through and pass-through paths.

routing can provide higher bandwidth for adversarial traffic patterns. For example, for a given traffic pattern (e.g., CPU0 to HMC1 in Fig. 9), there is a single, minimal path but if this path is congested, there are three non-minimal paths available by routing through an intermediate HMC. By exploiting the non-minimal paths, significantly more bandwidth can be provided between a CPU and an HMC for an adversarial traffic pattern. While this increases the hop count, it does not necessarily increase the latency since reduced congestion along non-minimal path can lower overall latency. Non-minimal routing has been used in large-scale networks to exploit path diversity and provide higher network throughput [16], and we adopt Universal Globally Adaptive Routing (UGAL) [17] to provide path diversity. The UGAL routing algorithm compares the congestion of a minimal path and a randomly chosen non-minimal path to determine whether to route a packet minimally or non-minimally.

C. Reducing Latency with Pass-through Microarchitecture

In our memory-centric network organization, the removal of CPU-to-CPU channels increases the CPU-to-CPU communication latency. The latency (T) of a packet through an HMC network can be summarized as

$$\begin{aligned} T &= T_s + HT_w + HT_r \\ &= T_s + HT_w + H(T_{SerDes} + T_{r\_NoC} + T_{w\_NoC}) \end{aligned}$$

where T_s is the serialization latency, T_w is the channel link latency, and $HT_w + HT_r$ is the header latency (H is the hop count and T_r is the per-hop latency through an HMC switch). By having to route through intermediate HMCs, T_s does not change. However, the additional hop count increases the aggregate channel link latency and the header latency proportionally – which includes the serialization/deserialization latency through the SerDes (T_{SerDes}), and the *intra-HMC*

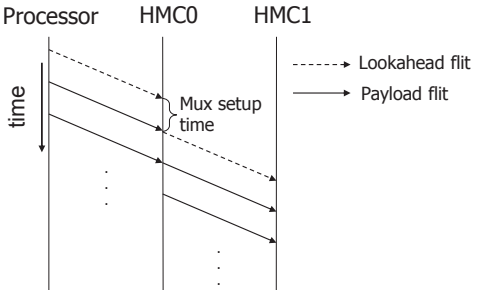


Fig. 12. Timeline showing the lookahead signal being sent prior to the payload flits for pass-through.

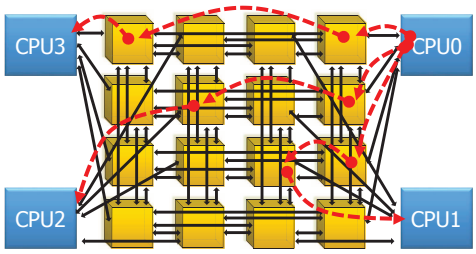


Fig. 13. Different pass-through paths from CPU0 to other CPUs highlighted with dashed arrows.

network latency which includes the on-chip router delay (T_{r_NoC}), and the on-chip wire delay (T_{w_NoC}). For latency-sensitive workloads, this additional latency can impact overall performance. To overcome this limitation, we propose a pass-through architecture that minimizes this latency through a combination of both circuit and microarchitecture techniques. In the pass-through architecture, we attempt to minimize the latency through intermediate HMCs (i.e., T_{SerDes}, T_{r_NoC}, and T_{w_NoC}) by forwarding the packet from the input I/O port directly to the output I/O port.

To minimize T_{w_NoC}, we exploit the intra-HMC network and its floorplan. As described earlier in Section II-B, the intra-HMC network is a 24×24 network. However, there is limited connectivity between the 24 ports since the memory controllers do not communicate with each other. Also, with the large size of an HMC logic die ($\approx 100mm^2$), it will be difficult to have a centralized crossbar used for the intra-HMC network as the I/O ports will be distributed throughout the die and thus, require long wires to route the data to a centralized crossbar and distributing the output to the various locations. The study of optimal intra-HMC network is outside the scope of this work but for our evaluation, we assume a concentrated mesh [18] intra-HMC network as shown in Fig. 10. [5] We assume a 6-way concentration where, within each concentration, there are four vault memory controller nodes and two I/O ports. By exploiting this floorplan and intra-HMC network organization which co-locates the I/O ports that are near each other, we are able to minimize the T_{w_NoC} component and support pass-through with minimal latency overhead.

Figure 11 shows the block diagram of the pass-through microarchitecture. High-speed serial links are used to connect different nodes and flits that arrive at a node through an input

[5] With a small number of "nodes", the topology can be viewed as 2×2 mesh or a 4 node ring.

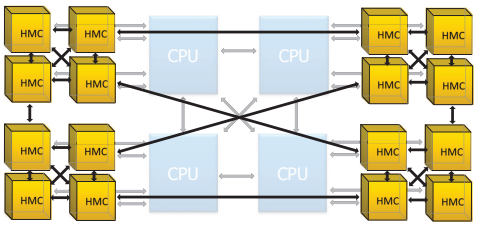


Fig. 14. A hybrid organization that combines the processor-centric network and the memory-centric network.

port of a link are deserialized to be properly processed within the node. Because the input port and the rest of the node is in a different clock domain, a synchronizer is needed. We assume that the deserialization and the synchronization process takes two node-clock (or HMC clock) cycles. Normal flits go through on-chip network datapaths, such as input buffers, crossbars, and arbitration, and then, are serialized at the output port before being transmitted to another node. The serialization process is assumed to take two node-clock cycles. Header flits need additional routing and virtual channel allocation steps. In order to minimize the latency within a node, we added datapath lines between two physically adjacent links. Good candidates of the adjacent links are the ones in the same concentration. We insert an output multiplexer (MUX) before the serializer of an output port and choose flits from either the output buffer of the output port or the synchronizer output of the adjacent input link, which bypasses regular datapath.

As the output MUX needs setup time, for pass-through packets, a lookahead flit is sent prior to the payload packet as shown in Fig. 12 to indicate that the following packet needs to be sent via pass-through logic to the co-located I/O port. The lookahead information can be embedded within a normal flit that proceeds the packets to be passed-through as well. In order to guarantee that the packet interleaving does not occur, the reservation of the pass-through path succeeds only if the following conditions are met:

- The virtual channel (VC) buffer of the payload packet is empty.
- There is enough credit for the VC buffer at the downstream HMC's router.
- Any prior packet that used this VC buffer is drained out of current router.

If any of the conditions is not met, the following payload packet goes through the deserializer and buffered at the input buffer. Thus, pass-through can be effective only when channel loads are low and, to maximize the use of pass-through paths, separate sets of VCs are required. If there is no lookahead flit, the output signal defaults to being received from the serializer.

Since the goal of pass-through is to minimize latency (in particular zero-load latency), we provide only single pass-through between each source CPU - destination CPU combination. The physical logic for pass-through can be combined (i.e., a single source CPU has path-through logic to two output ports) but because of the circuit complexity of supporting such pass-through design, we assume a separate pass-through logic for each source CPU-destination CPU combination. An example of the different pass-through paths for different source-destination combination is shown in Fig. 13 for CPU0. By using the distributed topology, each HMC input channel can be used for different pass-through path and destined to different destination CPU. In addition, since each source-destination pair has its own pass-through path within the intra-HMC network, the output I/O port can be placed adjacent to the input I/O port in the intra-HMC network (Fig. 10).

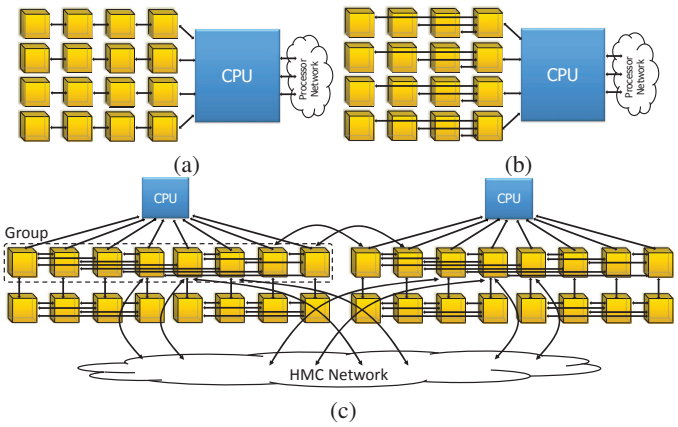


Fig. 15. Scaling HMC networks for (a) the processor-centric network, (b) the hybrid network, and (c) the memory-centric network. Inter-cluster channels are not shown for the hybrid network for simplicity.

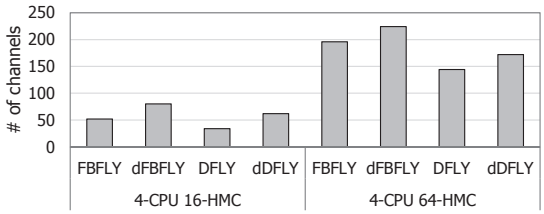


Fig. 16. Cost comparison in terms of the total number of channels for flattened butterfly- and dragonfly-based memory-centric networks.

D. Hybrid Network

In addition to the processor-centric and memory-centric networks, a *hybrid* organization is possible as shown in Fig. 14 for a 4CPU-16HMC network. Similar to a processor-centric network, the CPUs are directly connected to each other. In addition, similar to memory-centric network (MCN), the HMCs form a local network and some of the HMCs are directly connected to remote HMCs as well. Similar to MCN, the hybrid network can also benefit from adaptive routing that was described in the previous section.

E. Scaling the HMC Network

As with any network, the HMC network needs to scale appropriately as the number of HMCs increases in the system. Examples of scaling different HMC networks are shown in Fig. 15. For the PCN, additional HMCs are connected in a daisy-chain organization (Fig. 15(a)) while the CPUs are still connected to each other. To minimize the local HMC access latency, pass-through can be leveraged along the daisy-chain. The hybrid network can provide better connectivity among the local HMCs (Fig. 15(b)). The inter-cluster channels are not

TABLE I
EVALUATED CONFIGURATIONS FOR THE 4CPU-64HMC SYSTEM

Abbreviation	Configuration
PCN_MIN	Processor-centric network with minimal routing
PCN_MIN-p	PCN_MIN with pass-through
Hybrid_MIN	Hybrid network with minimal routing
Hybrid_UGAL	Hybrid network with with adaptive routing
MCN_MIN	Memory-centric network with minimal routing
MCN_UGAL	Memory-centric network with adaptive routing
MCN_MIN-p	MCN_MIN with pass-through
MCN_UGAL-p	MCN_UGAL with pass-through

TABLE II
PROCESSOR CONFIGURATION

Parameter	Value
Core	64 Out-of-Order cores @ 4 GHz Issue width: 4, ROB size: 64
L1 I/D cache	64 KB, 4-way, 1-cycle latency
L2 cache	Private 256 KB, 16-way, 10-cycle latency
Cache coherence	Directory-based MOESI
Cache line size	128 B
Directory	Full directory, 1-cycle, 4 per processor

TABLE III
HMC CONFIGURATION

Parameter	Value
HMC organization	8 layers × 16 vaults
HMC memory size	4 GB
Memory scheduler	FR-FCFS [19]
DRAM timing	tCK=1.25ns, tRP=11, tCCD=4, tRCD=11, tCL=11, tWR=12, tRAS=22

TABLE IV
EVALUATED SPLASH-2 WORKLOADS

Name	Input problem size	L2 MPKI
Barnes	64K particles	2.53
Cholesky	d750.O	1.28
FFT	1M data points	2.31
FMM	16K particles	0.73
LU	512 × 512 matrix	9.76
Ocean	258 × 258 grids	10.19
Radix	32M integers	4.39
Raytrace	car	6.69
Water-Sp	256 molecules	0.32

shown in Fig. 15(b), but the same inter-cluster connectivity among HMCs shown in Fig. 14 are assumed for the hybrid network. Figure 15(c) shows an example of how memory-centric networks can be scaled. In the 16HMC system, dFBFLY provided the best performance result because of the lower diameter as we discuss in Section V. However, as we show in Fig. 16, the dFBFLY does not necessarily scale well with larger network size due to the larger amount of channels required to provide all of the connectivity. The dMESH topology suffers from higher hop count. Thus, for larger networks, we focus on leveraging the dDFLY topology.

IV. EVALUATION METHODOLOGY

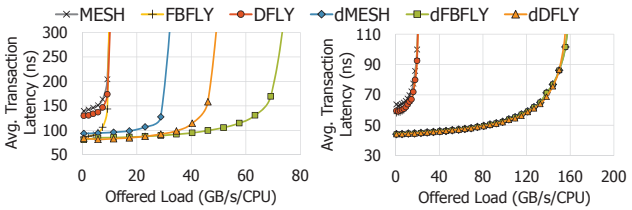
We evaluated the design space of system interconnects with both synthetic traffic and real workloads. The evaluated configurations are summarized in Table I and we evaluated different size networks, including 4CPU-16HMC and 4CPU-64HMC systems. For synthetic traffic evaluation, the different topologies were implemented in a cycle-accurate interconnection network simulator [20] and we evaluated four different traffic patterns – local HMC, CPU-to-CPU, remote HMC, and adversarial traffic patterns. For local HMC traffic pattern which is representative of a common case, each CPU socket sends requests to a randomly selected HMC within its local HMC and the HMC generates a reply packet. For CPU-to-CPU traffic pattern which emphasizes coherence traffic, the request-reply traffic pattern occurs between the different CPU sockets. With a remote HMC traffic pattern, CPU sends requests to access a random *remote* HMC. To model a cache coherent system, the request is first sent to the corresponding remote CPU socket, and then the request is forwarded to the target remote HMC. The remote HMC sends back a reply to the requester CPU but is not necessarily routed through the remote processor as there can be alternative or shorter paths to the requester depending on the network topology – such as the hybrid or memory-centric network. We also evaluated performance for an adversarial traffic pattern where each CPU only sends requests to one of its local HMCs to create a hotspot traffic. [6] For all traffic patterns, the destination vault within the destination HMC is randomly chosen. We varied the request injection rate of the processors and plot offered load vs. average transaction latency, which measures the time from the generation of a request to the arrival of the reply. We use 1-to-1 ratio for read and write requests. A short packet consists of a single flit and a long packet consists of 8 (7) flits for MCN (PCN or hybrid network) with the flit size being 20 (25) bytes.

The real workload evaluation was done using detailed cycle-accurate simulator with the core model from McSimA+ [21], cache model from GEMS [22], and Booksim [20] network simulator. We modified a coherency protocol of GEMS such that detailed DRAM scheduling is done by the controllers within the HMC chip and, in case of a remote HMC access, the reply from the HMC can be directly sent to the requester without being necessarily routed through the home processor node.

Since the system that we evaluate is a NUMA (Non-Uniform Memory Access) system, we use a simple **first-touch** policy [23] where a newly allocated page is located at a local HMC of the CPU whose thread accesses it first and the page is not migrated thereafter. We used RW:CLH:VL:CB:CLL:LY:BY memory address mapping scheme. (RW:Row, VL:Vault, CB:Cube, LY:Layer, CLH:Column High, CLL:Column Low, BY:Byte Offset)

Simulated system configuration for the processors and HMCs are shown in Table II and Table III. We assumed 14 VCs channels which were partitioned across the 3 message classes. The VC usage to avoid routing deadlock is similar to prior work [15] for minimal and non-minaiml routing. For networks with pass-through, one VC for each message class is

[6] An adversarial traffic pattern where the CPU sends requests to a single *remote* HMC will create more traffic but such traffic pattern is highly unlikely.



(a) CPU-to-CPU (b) CPU-Local HMC

Fig. 17. Synthetic traffic evaluation of different MCN topologies for the 4CPU-16HMC system.

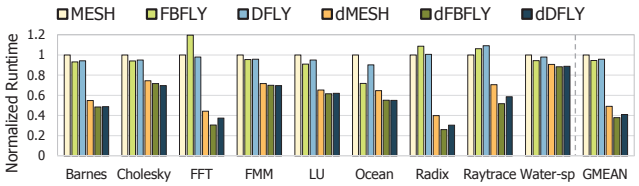


Fig. 18. Performance of different memory-centric networks for SPLASH-2 workload in the 4CPU-16HMC systems. All runtime values are normalized to MESH.

reserved for pass-through packets. Each VC buffer size is 500 bytes. We assume 1.25 GHz router frequency and 3.2ns SerDes latency (1.6 ns for each of serialization and deserialization).

In our evaluation, a processor in an MCN has 8 channels, while a processor in a PCN and a hybrid network has 7 channels. To keep the processor bandwidth constant, we assumed that a channel in an MCN operates at 10 Gbps per lane (using 4× router frequency with DDR), while a channel in a PCN operates at a higher 12.5 Gbps per lane (5× router frequency with DDR) to compensate for the fewer processor channels. All channels consist of 16 lanes (or differential pairs) in each direction. Across the different network topologies for a given system size, we kept the bisection channel count constant for fair comparison except for the PCN, which does not have HMC-to-HMC connectivity by definition, and thus has lower bisection bandwidth. Although this can be an unfair comparison, the purpose of including the PCN evaluation is to assess the benefits of proposed designs over traditional system interconnect which simply replaces DIMMs with HMCs. HMCs in the evaluated topologies have different number of channels, but adding more HMC channels does not result in higher performance as the bottleneck is the processor I/O bandwidth, which is held nearly constant across the different topologies.

For real workload evaluation, we used SPLASH-2 [12] benchmark suite with input problem size given in Table IV. [7] We evaluated system energy consumption using McPAT [24] for the CPU, CACTI-3DD [25] for the HMC's DRAM, and our interconnect energy model, which includes energy consumption by high-speed I/O channels, described as follows:

$$\begin{aligned}\text{Network Energy} =& E_{bit,real\ pkt} \times \sum_{i=1}^{n_c} D_{i,real} \\ &+ E_{bit,idle\ pkt} \times \sum_{i=1}^{n_c} D_{i,idle} \\ &+ E_{bit,router} \times \sum_{i=1}^{n_r} D_{i,router},\end{aligned}$$

where $E_{bit,real\ pkt}$ ($E_{bit,idle\ pkt}$) is the sum of Tx and Rx energy per bit for real (idle) packet [8] and $E_{bit,router}$ is per-bit energy consumed by router core to process packets. n_c and n_r denote the number of channels and routers in the network, respectively, and, $D_{i,real}$ ($D_{i,idle}$) are the total amount of real (idle) packet data transferred in bits, and $D_{i,router}$ is the amount of data in bits processed by router i. We assumed $E_{bit,real\ pkt}$ to be 2.0 pJ/bit, $E_{bit,idle\ pkt}$ to 1.5 pJ/bit based on the state-of-the-art designs [26], [27] and $E_{bit,idle\ pkt}$ to be 1 pJ/bit based on [4]. For MIN routing, some of the HMC-to-HMC channels are not utilized and we assumed those links are power-gated and do not consume any power.

[7] Among the SPLASH-2 applications, `volrend` and `radiosity` were not available due to the limitations of simulation infrastructure.

[8] High-speed signaling requires sending/receiving idle packets over the channels when there is no data to communicate.

V. Evaluation

Our evaluation results with synthetic traffic and real workload show that distributor-based MCN provides high network throughput for various traffic patterns while resulting in low latency by using pass-through paths. Hybrid network improved network throughput compared to PCN, but its local HMC bandwidth is limited as some processor channels are dedicated to the processor network. Non-minimal routing capability can also improved throughput for adversarial traffic pattern by leveraging path diversity.

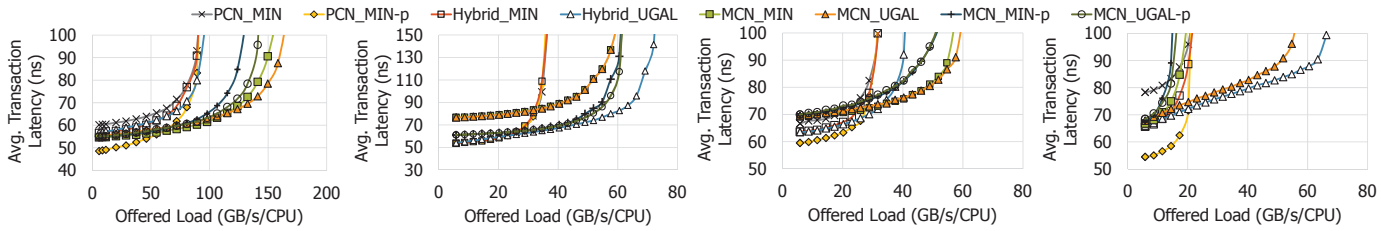
A. Impact of Distributor-based Network

We first compare the impact of using *distributor*-based network, compared with the topology without a distributor. The results for synthetic traffic patterns are shown in Fig. 17 and real workloads in Fig. 18 for a 4CPU-16HMC system. Overall, using distribution reduced zero-load latency since the network diameter was further reduced. In addition, better utilization of CPU channels significantly improved overall throughput. Across the different topologies, dFBFLY provided the best performance for the different traffic patterns. Similar trend is also shown in real workloads (Fig. 18). Using distribution reduced the runtime by up to 40% across the different topologies. Although dFBFLY provided the best performance, due to its limited scalability discussed in Section III-E, we assumed distributor-based dragonfly (dDFLY) topology for the different 4CPU-64HMC MCN networks in the following sections.

B. Synthetic Traffic

The synthetic traffic evaluation results for the 4CPU-64HMC system are shown in Fig. 19. Overall, the synthetic traffic evaluation shows that the MCN significantly improved the local HMC bandwidth compared to the hybrid network while providing comparable throughput with the hybrid network for CPU-to-CPU, remote HMC and the adversarial traffic pattern. Both the MCN and the hybrid network outperformed the PCN for different traffic patterns evaluated in this work.

Local HMC traffic performance is shown in Fig. 19(a). By using our proposed pass-through logic, the zero-load latency of the PCN was significantly reduced by 25% compared to the baseline PCN. Other networks resulted in slightly higher zero-load latency. With the PCN and hybrid network, only 4



(a) CPU-Local HMC (b) CPU-to-CPU (c) CPU-Remote HMC (via remote CPU) (d) Adversarial traffic

Fig. 19. Synthetic traffic results for the 4CPU-64HMC system.

out of 7 channels can be used to access local HMCs, so the throughput is limited to 100 GB/s. Considering the second generation HMC prototype provides 160 GB/s with a single HMC [4], the actual bandwidth available to the CPU is fairly limited and the high bandwidth benefit of the HMC cannot be fully utilized with the PCN or hybrid network. However, with the MCN, all processor bandwidth can be used to access local HMCs and the throughput is improved by up to 1.5× for MCN_UGAL.

Figure 19(b) shows the CPU-to-CPU traffic pattern performance. The hybrid network with MIN routing provided no benefit over the PCNs in this traffic pattern as inter-cluster channels were not utilized, but UGAL routing significantly improved throughput by utilizing those channels. On the other hand, MCNs, without pass-through, suffered from 42% higher zero-load latency compared to the PCN as CPU-to-CPU traffic took additional hops through intermediate HMCs. Employing the pass-through significantly reduced this gap by making per-hop latency shorter for the CPU-to-CPU traffic, resulting in only 13% increase in zero-load latency over the PCN and hybrid network. Even though zero-load latency for the CPU-to-CPU traffic was increased with the MCN, as the traffic load increases, the MCN can provide lower latency due to its flexible use of processor bandwidth compared to the PCN. The MCN provided comparable throughput without pass-through, but adopting pass-through degraded the throughput as fewer VCs were available for normal packets as the total number of VCs was kept constant. For the same traffic pattern, the throughput of the MCNs was improved by 62% compared to the PCNs as more channels were available for CPU-to-CPU packets. With the hybrid network, MIN routing did not result in any improvement over the PCN, but UGAL routing provided 94% higher throughput.

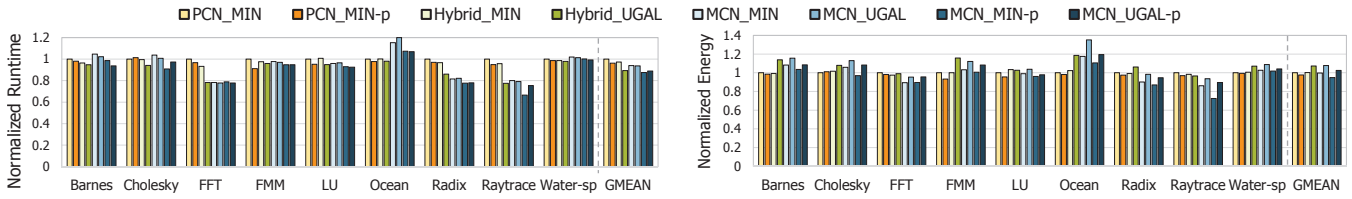
With the remote HMC traffic (Fig. 19(c)), the zero-load latency of the MCN is higher than that of other networks, but it provides higher throughput as the processor bandwidth is more efficiently utilized. For the adversarial traffic pattern (Fig. 19(d)), MIN routing provides very low throughput across the different topologies as path diversity is not exploited and non-minimal paths are not utilized. However, with UGAL and adaptive non-minimal routing, the throughput of the MCN and hybrid network was significantly improves over the PCN by 2.8× and 3.4×, respectively.

C. Real Workload

SPLASH-2 results for different networks are shown in Fig. 20(a). We normalize runtime and energy values to those of the baseline (PCN_MIN) configuration. The PCN_MIN-p, due to its lower local HMC access latency with pass-through, reduced runtime by 3.7% on average. Compared to the baseline, different MCN configurations significantly reduced average runtime. With MCN_MIN-p, the performance was improved by 12% on average and up to 33% for `Raytrace`. For other memory bandwidth intensive benchmarks such as `FFT`, and `Radix`, MCN_MIN-p reduced the runtime by 21% and 22%, respectively. Figure 21 shows the average packet latency for CPU-to-CPU traffic and local HMC traffic along with the fraction of packets for each traffic pattern in the overall traffic. For `Raytrace`, the average CPU-to-CPU latency slightly increased with the MCN compared to the PCN (Fig. 21(a)). However, due to higher local HMC I/O bandwidth, the average local HMC latency was significantly reduced and resulted in improved performance. ***The large HMC I/O bandwidth in the MCN results from dedicating all of the processor bandwidth to HMC bandwidth – while in the PCN, some of the processor bandwidth is dedicated to CPU-to-CPU communication.*** For `FFT`, the CPU-to-CPU latency for large packets was higher with the MCN_MIN-p compared to other networks as some VCs were reserved for pass-through paths and were not available for normal packets, but the short control packets experienced 85% lower latency in the MCN_MIN-p compared to PCN_MIN-p as CPU channels were better load-balanced in the MCN and short packets were able to make better use of pass-through paths. Since the control packets dominated the CPU-to-CPU traffic and local HMC access latency was lowered, the performance was significantly improved. The performance of `Radix` was improved for the same reason.

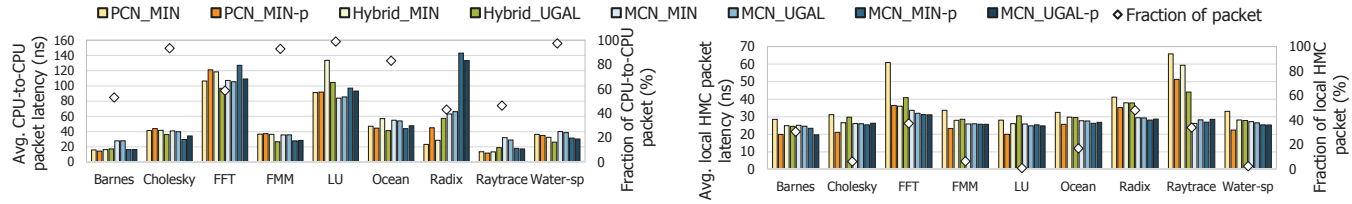
In the workloads that we evaluated, unlike synthetic workloads, there was not sufficient bandwidth usage to show the benefits of UGAL over MIN routing for the MCN network. However, for workloads with more memory bandwidth requirements, we expect UGAL routing to provide performance improvement over MIN routing. In comparison, UGAL routing was very beneficial for the hybrid network when the CPU-to-CPU traffic load was high as the packets were routed non-minimally and resulted in lower overall packet latency. As a result, Hybrid_UGAL reduced the runtime by 8% on average compared to Hybrid_MIN.

For the other non-intensive bandwidth benchmarks, different topologies showed very little difference in performance. MCN increased the CPU-to-CPU latency in some cases, but by using pass-through logic, the latency was reduced and comparable to the baseline. Figure 20(b) shows total system energy result. The energy consumption by the interconnect was increased with the MCN and hybrid network since more links were used compared to the PCN. However, the overall system energy was reduced as the runtime was reduced and



(a) Normalized runtime (b) Normalized energy

Fig. 20. Performance and energy result for SPLASH-2 workload on the 4-CPU 64-HMC system.



(a) CPU-to-CPU packet (b) local HMC packet

Fig. 21. Packet latency and fraction of packet for (a) CPU-to-CPU packet and (b) CPU-local HMC packet.

CPU energy consumption was lowered as a result, which represents a higher fraction than interconnect energy. Overall, the MCN_MIN-p reduced energy consumption by 5.3% (2.8%) compared to the PCN (PCN_MIN-p). However, with UGAL, more links were active than with MIN, and the system energy consumption increased. Hybrid_UGAL, while providing comparable performance to MCN_MIN-p, resulted in 7.3% higher energy.

VI. Related Work

Different organizations for extending a single HMC to multiple HMCs, including a conventional mesh and a ring topology, were presented [3] but these approaches will not scale well since it results in higher hop count and thus, increased latency and energy. High-radix topologies [15] have been recently proposed to reduce the network diameter. We also attempted to minimize the network diameter but given the unique constraints of an HMC network (i.e., a larger number of "routers" or HMCs compared with a smaller number of nodes or processors), previously proposed topologies are not directly applicable. In contrast to the Intel QPI or HyperTransport which propose multi-hop networks for more than 4 CPU sockets, the SPARC T5 [28] provides a fully-connected processor interconnect with one hop access to up to 8 sockets. However, this organization is still a processor-centric network and in this work, we propose an alternative memory-centric network organization that leverages HMCs.

Alternative memory architectures such as fully-buffered DIMM (FB-DIMM) [29] have been proposed to replace conventional bus-based parallel memory interfaces with high-speed serial interfaces. One disadvantage of FB-DIMM is the additional latency and power introduced by having to send data through intermediate advanced memory buffers (AMBs). We try to avoid this disadvantage by introducing a distributor-based network that reduces the memory network diameter. In addition, similar to FB-DIMM, we also propose using pass-through logic to reduce latency but the logic differs significantly since the AMB only contains two inputs (northbound and southbound) for FB-DIMM while the number of ports is much higher in HMCs. To populate more DIMMs, buffer-on-board can be used [30]; however, DIMMs are essentially connected via shared buses.

Memory organizations with alternative signaling technologies have been proposed to overcome some of the limitations of the traditional DIMM structure. A DIMM tree structure [31] has been proposed to provide better scalability while leveraging multiband radio-frequency interconnects to overcome the limited scalability of multi-drop buses. Udipi et al. [32] proposed using silicon photonics to interconnect a CPU with multiple 3D-stacked memory chips using a ring network. In our work, we do not explore alternative interconnect technologies but assume conventional, electrical signaling. It remains to be seen how alternative signaling technologies impacts the design of memory-centric network organization.

VII. Conclusions

In this work, we explored a system interconnect design that leverages the hybrid memory cubes (HMCs). We showed how, instead of a *processor*-centric approach, an alternative *memory*-centric organization can better exploit the advantages provided by the HMC memory system. We proposed a distributor-based network that effectively utilized both the processor I/O bandwidth and the HMC I/O bandwidth. To provide better bandwidth flexibility, we showed how non-minimal adaptive routing needs to be leveraged and, to reduce the latency, we described how the pass-through of the intermediate HMC router can be exploited. With our proposed architecture, the throughput of a memory-centric network is able to exceed the throughput of a processor-centric network and hybrid network by 50% on uniform random local HMC traffic pattern. For an adversarial traffic pattern, the improvement is by 2.8× over processor-centric network. For real workloads, the proposed network (MCN_MIN-p) improved the performance by up to 33% (12% on average) while reducing the energy consumption by 5.3% on average.

In this work, we focused on a system interconnect for a system within a single board. However, as system size increases, it remains to be seen how to efficiently scale the

system interconnect with HMCs. In addition to the network design itself, the network will also impact other aspects of the memory system design, including optimal cache coherence, page migration to exploit the difference between local and remote HMCs, and memory scheduling. We plan to explore how the proposed interconnect architecture impacts these other aspects.

ACKNOWLEDGMENTS

We thank the reviewers for their comments. This work is supported in part by SK Hynix. John Kim is supported in part by the MSIP, Korea, under the ITRC support program supervised by the NIPA (NIPA-2013-H0301-13-1011) and in part by Basic Science Research Program through the NRF funded by the MSIP (NRF-2011-0015039). Jung Ho Ahn is partially supported by the Basic Science Research Program through the NRF funded by the MSIP (2012R1A1B4003447).

REFERENCES

[1] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for CMP scaling," in *ISCA'09*.

[2] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, Mar. 1995.

[3] J. T. Pawlowski, "Hybrid memory cube (HMC)," *Hot Chips 23*, 2011.

[4] G. Sandhu, "DRAM scaling and bandwidth challenges," in *NSF Workshop on Emerging Technologies for Interconnects*, 2012.

[5] "Hybrid Memory Cube Specification 1.0," [Online]. Available: http://www.hybridmemorycube.org/, Hybrid Memory Cube Consortium, 2013.

[6] D. Ziakas, A. Baum, R. Maddox, and R. Safranek, "Intel QuickPath Interconnect architectural features supporting scalable system architectures," in *HOTI'10*.

[7] "HyperTransport I/O technology overview," The HyperTransport Consortium, Tech. Rep., June 2004.

[8] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA: Morgan Kaufmann, 2004.

[9] S. Scott, D. Abts, J. Kim, and W. J. Dally, "The BlackWidow high-radix clos network," in *ISCA'06*.

[10] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini system interconnect," in *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*.

[11] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 Tb/s 6x4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 4, pp. 757–766, 2011.

[12] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," in *ISCA'95*.

[13] B. L. Jacob, *The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[14] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," in *ISCA'07*.

[15] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Cost-efficient dragonfly topology for large-scale systems," *IEEE Micro*, vol. 29, no. 1, pp. 33–40, 2009.

[16] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: a scalable HPC system based on a Dragonfly network," in *SC'12*.

[17] A. Singh, "Load-balanced routing in interconnection networks," Ph.D. dissertation, Stanford University, 2005.

[18] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *SC'06*.

[19] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *ISCA'00*.

[20] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *ISPASS'13*.

[21] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+Simulation and Detailed Microarchitecture Modeling," in *ISPASS'13*.

[22] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, Nov. 2005.

[23] J. Antony, P. P. Janes, and A. P. Rendell, "Exploring thread and memory placement on NUMA architectures: Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport," in *HiPC'06*.

[24] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing," *ACM Transactions on Architecture and Code Optimization*, vol. 10, no. 1, 2013.

[25] K. Chen, S. Li, N. Muralimanohar, J. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *DATE'12*.

[26] K. Fukuda, H. Yamashita, G. Ono, R. Nemoto, E. Suzuki, T. Takemoto, F. Yuki, and T. Saito, "A 12.3mW 12.5Gb/s complete transceiver in 65nm CMOS," in *ISSCC'10*.

[27] J. Poulton, R. Palmer, A. M. Fuller, T. Greer, J. Eyles, W. J. Dally, and M. Horowitz, "A 14-mW 6.25-Gb/s Transceiver in 90-nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 12, 2007.

[28] J. Feehrer, S. Jairath, P. Loewenstein, R. Sivaramakrishnan, D. Smentek, S. Turullols, and A. Vahidsafa, "The Oracle Sparc T5 16-Core Processor Scales to Eight Sockets," *IEEE Micro*, vol. 33, no. 2, pp. 48–57, 2013.

[29] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, "Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling," in *HPCA'07*.

[30] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer-on-board memory systems," in *ISCA'12*.

[31] K. Therdsteerasukdi, G.-S. Byun, J. Ir, G. Reinman, J. Cong, and M.-C. F. Chang, "Utilizing Radio-Frequency Interconnect for a Many-DIMM DRAM System," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 2, no. 2, pp. 210–227, 2012.

[32] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "Combining Memory and a Controller with Photonics through 3D-Stacking to Enable Scalable and Energy-Efficient Systems," in *ISCA'11*.