2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10071005

DIMM-Link: Enabling Efficient Inter-DIMM Communication for Near-Memory Processing

Zhe Zhou*[1,2,3], Cong Li*[1,3], Fan Yang[4], Guangyu Sun†[1,3]
[1]*School of Integrated Circuits*, [2]*School of Computer Science*, *Peking University*
[3] *Beijing Advanced Innovation Center for Integrated Circuits*
[4] *School of Computer Science, Nankai University*
{*zhou.zhe, leesou, gsun*}*@pku.edu.cn*, *yangf@nbjl.nankai.edu.cn*

***Abstract*—DIMM-based near-memory processing architectures (DIMM-NMP) have received growing interest from both academia and industry. They have the advantages of large memory capacity, low manufacturing cost, high flexibility, compatible form factor, etc. However, inter-DIMM communication (IDC) has become a critical obstacle for generic DIMM-NMP architectures because it involves costly forwarding transactions through the host CPU. Recent research has demonstrated that, for many applications, the overhead induced by IDC may even offset the performance and energy benefits of near-memory processing.**

To tackle this problem, we propose DIMM-Link, which enables high-performance IDC in DIMM-NMP architectures and supports seamless integration with existing host memory systems. It adopts bidirectional external data links to connect DIMMs, via which point-to-point communication and inter-DIMM broadcast are efficiently supported in a packet-routing way. We present the full-stack design of DIMM-Link, including the hardware architecture, interconnect protocol, system organization, routing mechanisms, optimization strategies, etc. Comprehensive experiments on typical data-intensive tasks demonstrate that the DIMM-Link-equipped NMP system can achieve a 5.93× average speedup over the 16-core CPU baseline. Compared to other IDC methods, DIMM-Link outperforms MCN, AIM, and ABC-DIMM by 2.42×, 1.87×, and 1.77×, respectively. More importantly, DIMM-Link fully considers the implementation feasibility and system integration constraints, which are critical for designing NMP architectures based on modern DDR4/DDR5 DIMMs.

I. Introduction

DIMM (Dual-Inline Memory Module)-based Near-Memory Processing architectures (DIMM-NMP) have been proposed for many years to address the huge performance gap between the CPU and main memory. By placing processing units near DRAMs and offloading memory-intensive operations to the NMP cores, we can harness the high aggregated memory bandwidth and mitigate energy-consuming off-chip/package data movement. In the big-data era, the increasing demands for data-intensive workloads also accelerate the adoption of DIMM-NMP architectures in scenarios like graph processing [14], [77], genome analysis [11], [39], personalized recommendation [4], [44], [45], [52], [68], and machine learning [48], [58], [89], etc.

Compared to the HBM/HMC-based near-memory processing counterparts [1], [28], [51], [88], [91], DIMM-NMP designs have larger memory capacity, lower manufacturing cost, higher flexibility, and a compatible form factor for drop-in replacement [5], [44]. Thus, they have the potential to become commodity devices for massive deployment in data centers. Consequently, UPMEM and Samsung have presented commodity [32] and conceptional [49], [70] DIMM-NMP products in recent years. Evaluations with real applications demonstrate that DIMM-NMP can improve performance and energy efficiency by several times [32], [70].

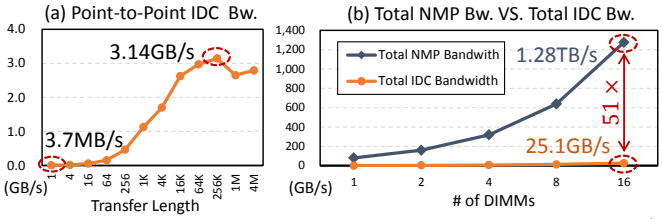


Fig. 1. IDC Performance Exploration on the Real DIMM-NMP Platform[1].

Though providing high intra-DIMM bandwidth, DIMM-NMP cannot directly support inter-DIMM communication (IDC), which is, however, required by many data-intensive tasks (e.g., for graph processing, a DIMM usually needs to access the neighbor vertices stored in other DIMMs). Therefore, many DIMM-NMP architectures only focus on IDC-free tasks (e.g., personalized recommendations [44], [52], [68]). Other generic approaches [3], [9], [32], [45] count on the host CPU to forward inter-DIMM transactions. Specifically, to transfer data among DIMMs, the host CPU has to load data from the source DIMM to its cache hierarchy, then store it in the destination DIMM via the narrow memory bus. To demonstrate the inefficiency of such a method, we evaluate the IDC performance on the commodity UPMEM [32] platform[1]. As Figure 1 shows, the maximum point-to-point IDC bandwidth is merely 3.14GB/s, which can only be achieved in bulk data transfer. The 16-DIMM system provides 1.28TB/s NMP bandwidth, but can only provide roughly 25GB/s P2P IDC bandwidth (51× lower). Previous benchmarking [32], [76] has also addressed that such a CPU-forwarding mechanism suffers from low performance and weak scaling on many tasks involving frequent data exchange. Therefore, IDC has become a bottleneck for generic DIMM-NMP architectures.

* Co-first authors.
† Corresponding author.

[1]System configuration: 4 normal DDR4 DIMMs, 16 UPMEM DIMMs, 2048 DPUs@350MHz, 2× Xeon 4210R CPUs. SDK version: 2021.3.

A few recent works have tried to tackle such an IDC bottleneck. For instance, ABC-DIMM [76] leverages the multi-drop bus structure to enable data-broadcast inside a memory channel, which benefits some broadcast-dominant applications. However, it cannot help many other broadcast-unfriendly applications. Also, as it relies on the host to issue customized broadcast commands, ABC-DIMM involves costly modification to the host CPU hardware. Another work called AIM [11] adds a dedicated bus to connect all DIMMs. Different from ABC-DIMM, it allows the NMP cores to transfer data on the dedicated bus without any host CPU assistance. Both ABC-DIMM and AIM rely on the multi-drop bus structure, which has two major limitations. First, as the NMP cores have to compete for the shared bus, the point-to-point IDC bandwidth cannot scale as the number of DIMMs increases. Second, and more importantly, connecting many DIMMs (i.e., $\geq$ 4 DIMMs) using a multi-drop bus is only practical in the DDR2 and DDR3 eras [80]. For the high-frequency DDR4 and DDR5 DIMMs, the multi-drop bus structure is faced with severe timing and signal integrity issues [15], [46]. Therefore, the benefits of intra-channel broadcast are limited [76].

We argue that an ideal IDC mechanism for commodity DIMM-NMP architectures should possess the following characteristics. First, it should constrain the hardware innovations within the DIMM module and avoid modifying the host CPU.

Second, it should provide both point-to-point and broadcast communications to support generic near-memory processing applications. Third, it should achieve flexible and scalable IDC bandwidth as the number of DIMMs increases. Last, and perhaps most important, the implementation feasibility and integration constraints should be carefully considered to ensure practicability for modern computing systems.

To fulfill these goals, we propose a novel interconnect architecture for DIMM-NMP named DIMM-Link. Our main idea is to connect adjacent DIMMs in a system with high-speed external data links, via which both point-to-point communication and broadcast are efficiently supported in a packet-routing way. To this end, we present the full-stack design of DIMM-Link: we propose a DIMM-Link Bridge (DL-Bridge) to connect adjacent DIMMs with high-bandwidth SerDes links physically. The DL-Bridge is driven by the DL-Controllers that co-locate with the NMP cores. We design a packet-based interconnect protocol to standardize the transmission. Considering various system integration factors, we propose a "DL Group" concept to organize the DIMMs in groups and a hybrid routing mechanism to handle inter/intra-group packet transmission. To improve the performance further, we also introduce a polling proxy mechanism to mitigate the CPU-polling overhead and a distance-aware task-mapping strategy to improve the thread-data affinity. To summarize, we have made the following main contributions:

- We deeply review existing methods for inter-DIMM communication and analyze their limitations in respect of hardware modification, supported IDC modes, effective bandwidth, target applications, and practicability for modern DDR4/DDR5 DIMMs. (Section II-B)
- To overcome the limitations, we propose DIMM-Link, a novel interconnect solution to tackle the IDC bottleneck in DIMM-NMP architectures. We present detailed designs of hardware architecture, protocol stack, system organization, routing mechanisms, etc. (Section III)
- Based on DIMM-Link, we propose two optimization strategies to improve the IDC performance further, including a polling-proxy mechanism and a distance-aware task mapping strategy. (Section IV)

Experiments on various NMP tasks demonstrate that DIMM-Link achieves, on average, $5.93\times$ higher performance over the 16-core CPU baseline and outperforms MCN and AIM by $2.42\times$ and $1.87\times$, respectively. On broadcast-based tasks, DIMM-Link is also $1.77\times$ faster than ABC-DIMM.

II. Background & Motivation

In this section, we first briefly introduce different DIMM-NMP architectures and their execution flows. Then, we deeply review existing methods for inter-DIMM communication and analyze their limitations.

A. DIMM-NMP Architectures & Execution Flows

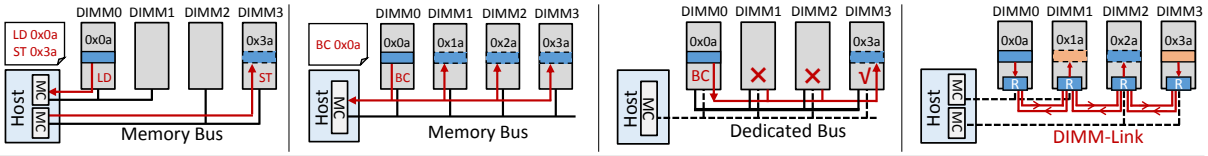
According to the locations of NMP cores, we can classify existing DIMM-NMP architectures into three main categories: (1) Processing near banks [17], [18], [32], which is adopted by the commodity UPMEM PIM-DIMM product [17]. (2) Processing in separated buffer chips [5], [39], and (3) Processing in a centralized buffer chip [3], [11], [44], [52], [76], [90], which is also adopted by Samsung's AxDIMM [49]. According to their execution flows, DIMM-NMP architectures can be classified as coarse-grained [3], [5], [11], [32], [76], or fine-grained [44], [49], [90] designs. With the coarse-grained flow, the host CPU first writes data and kernels to NMP DIMMs via the host memory controller (MC), after which the host MC hands over the control of the DRAMs to local MCs residing in DIMMs to execute the kernel. The host cannot access DRAMs during the kernels' execution to avoid conflicts. For fine-grained execution flow, the host CPU must send commands to drive the NMP core's execution. In this case, the NMP cores usually do not have complex local MCs, and instructions should have deterministic timing. *This paper focuses on the more general centralized-buffer architecture with a coarse-grained execution flow.*

B. Analysis of the Existing IDC Methods

Although DIMM-NMP can provide high intra-DIMM bandwidth, there is still a lack of an efficient inter-DIMM communication (IDC) mechanism, which is necessary for many data-intensive applications. Currently, there are mainly three ways to realize IDC in DIMM-NMP architectures, namely *CPU-forwarding, intra-channel broadcast, and dedicated bus*. We compare them in detail as follows:

CPU-Forwarding. MCN [3], UPMEM [32], and some other architectures [9], [39] rely on the host CPU to forward data among DIMMs. As shown in the second column of Table I, if NMP cores in DIMM-3 want to read data in DIMM-0, it

TABLE I
COMPARISONS OF INTER-DIMM COMMUNICATION METHODS.

IDC Methods	CPU-Forwarding [3], [32]	Intra-Channel Broadcast [76]	Dedicated Bus [11]	**DIMM-Link**
Illustration				
Hardware Modification	DIMM Modules	Host CPU, DIMM Modules	DIMM Modules	DIMM Modules
Supported IDC Modes	Point-to-Point	Broadcast	Point-to-Point	Point-to-Point & Broadcast
Maximum Bandwidth	$\#Channel \times \beta/2$	$\#DIMM \times \beta$	β	$\#Link \times \beta$
Target NMP Apps	IDC-infrequent Applications	Sparse Tensor Algebra	Computational Genomics	**Generic Applications**

writes the request to a local memory-mapped register. The host CPU periodically polls all these registers in DIMMs to get the IDC requests. After noticing DIMM-3's request, the host CPU reads the required data from DIMM-0 and writes it to DIMM-3 via the memory channels. Since the data copy occupies the channel twice, the theoretical bandwidth is $\#Channel \times \beta/2$, where β denotes the bandwidth of each memory channel.

This method has three limitations: (1) Forwarding is expensive, which involves the host CPU reading data into its cache hierarchy and then writing it back to the destination. It not only has considerable latency, but also affects the performance of concurrently-running applications that share the memory bandwidth [73]. (2) Since DIMMs share the memory channel by time-division multiplexing, they have to compete for the IDC bandwidth [76]. (3) The periodical CPU-polling occupies the scarce host CPU and memory bus resources [3], [39], even if no requests are issued. Therefore, in a real DIMM-NMP system adopting CPU-forwarding [17], the in-efficient IDC becomes the bottleneck of many applications [32].

Intra-Channel Broadcast. Instead of accelerating point-to-point communication, ABC-DIMM [76] uses the multi-drop bus to broadcast data inside a memory channel. As shown in the third column of Table I, when DIMMs 1-3 in the same channel all want to read the data at address `0x0a` in DIMM-0, the data is sent to all the DIMMs simultaneously via a `broadcast-read` command issued by the host CPU. In this case, the IDC bandwidth is $3\times$ higher than point-to-point transmission. However, ABC-DIMM has several limitations: (1) modern server systems equipping DDR4/DDR5 DIMMs cannot support many DIMMs in each channel due to signal integrity considerations [15]. As far as we know, the maximum number of DDR4 DIMMs per channel is merely three [37]. Therefore, the benefits of channel-wise broadcast are limited [76]. The inter-channel broadcast still relies on costly CPU-forwarding. (2) Many broadcast-unfriendly applications, such as BFS and K-means, cannot benefit from such a paradigm. (3) It modifies the host CPU to support the customized broadcast commands, significantly increasing the manufacturing and deployment costs.

Dedicated Bus. AIM [11] adopts a dedicated bus to augment the IDC capability. It connects all DIMMs with a dedicated bus and allows them to communicate without the assistance of host CPUs. An example is shown in the fourth column of Table I. When DIMM-3 wants to read data from DIMM-0, it broadcasts the commands via the dedicated bus. DIMM-0 snoops on the request and puts the data on the bus accordingly for DIMM-3 to fetch. Such a method eliminates the CPU-polling overhead since NMP cores control the data transmission independently. However, as addressed before, connecting all DIMMs with a multi-drop bus is extremely challenging for DDR4/DDR5 due to the timing and signal integrity issues [30], [53]. Moreover, such a design suffers from unscalable IDC bandwidth since all NMP cores compete for the shared bus [76]. Assume the bandwidth of the dedicated bus is also β, then the per-DIMM bandwidth is mere $\frac{\beta}{\#DIMM}$.

To summarize, although these methods have contributed valuable ideas for inter-DIMM communication, they are faced with at least one of the following challenges: (1) incurring costly modification to the host system, (2) working for specific applications, (3) providing unscalable IDC bandwidth, and (4) suffering from hardware implementation and system integration constraints. To tackle these challenges and design a potential commodity approach, we propose the DIMM-Link interconnect, which is introduced in the following sections.

III. DIMM-LINK

We propose to revamp DIMM-NMP with the DIMM-Link interconnect (the last column of Table I). DIMM-Link's main idea is to use external data links to connect adjacent DIMMs and use packet-based routing to transfer data among DIMMs flexibly. We overcome the challenges mentioned above and fulfill various design goals. First, the hardware modifications are constrained to DIMM modules to reduce deployment costs. Second, DIMM-Link supports both point-to-point and broadcast IDC to accelerate generic applications. Third, since data can be transmitted between adjacent DIMMs concurrently, the maximum IDC bandwidth scales with the number of data links. Last, implementation feasibility and system integration constraints are carefully considered.

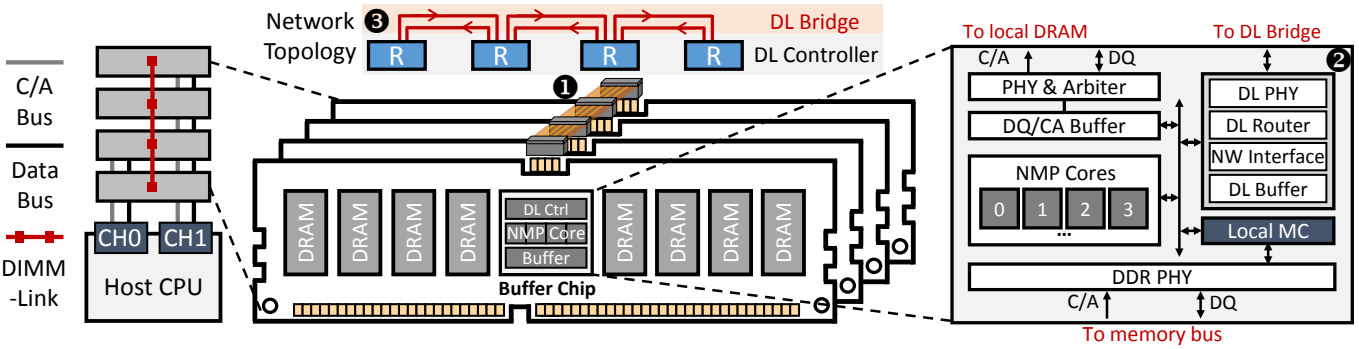


Fig. 2. DIMM-Link Architecture.

A. Architecture Overview

As illustrated in Figure 2, the DIMM-Link hardware consists of two main components: a DIMM-Link Bridge (DL-Bridge ❶) and multiple DIMM-Link Controllers (DL-Controller ❷). The DL-Bridge physically connects multiple DIMM modules via customized interfaces (the pins on the upper edge of each DIMM module, including data pins and power pins). Inside the DL-Bridge, there are bidirectional SerDes links to transfer data signals between adjacent DIMMs. In the centralized buffer chip of each DIMM, there is a DL-Controller to drive the DL-Bridge and handle the packet generation, routing and receiving, etc. As abstracted in ❸, N DIMMs connected via DIMM-Link ($N = 4$ in the figure) form a network topology containing $2\times(N-1)$ unidirectional links. Such a pluggable modular design facilitates the installation and maintenance of DIMM modules. Details about the DL-Bridge and DL-Controller are introduced as follows.

DL-Bridge. DL-Bridge is a specialized PCB board serving as the medium of IDC. Full-duplex bidirectional SerDes links are placed between the adjacent slots to transfer data signals. DL-Bridge and the routers in the connected DIMMs form a network that allows concurrent packet-based data transmission. Physically, DL-Bridge can be implemented using mature off-package SerDes links like GRS (Ground-Referenced Signalling) [69], which provides a high signal rate with low energy consumption. Moreover, since we only connect the adjacent DIMMs in a point-to-point way, the signal integrity is easy to guarantee. DL-Bridge is driven by the power pins of the DIMM-Link interface. Alternatively, we can use an external power cable to drive the DL-Bridge if necessary.

DL-Controller. The DL-Bridge only contains the necessary wires, slots, and transceiver logic to transfer signals. The packet generation, routing, and receiving are all handled by a DL-Controller residing in each DIMM's buffer chip (❷ in Figure 2). Apart from the physical interface (DL-PHY) to interact with the DL-Bridge, there are three key components in the DL-Controller: a DL-Router, an NW (Network)-Interface, and a DL-Buffer. When DL-Controller is invoked by a local request, the NW-Interface will packetize the request and put it into the DL-Buffer. The DL-Routers are responsible for routing these packets to the destination DIMM. A DL-Router also determines whether a packet has arrived at its destination. If so, the NW-Interface will decode the packet and feed the data to the right place. DL-Controller cooperates with the NMP cores, local memory controller, DL-Bridge, and host CPU to complete various types of IDC transactions. More details will be introduced in Section III-D.

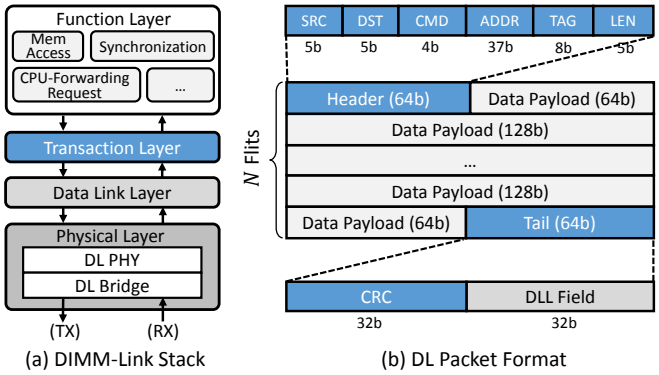


Fig. 3. DIMM-Link Protocol.

B. DIMM-Link Protocol

Like some other packet-based interconnects [2], [22], [40], [75], DIMM-Link also has a protocol to standardize the transmission. Figure 3-(a) shows that the whole protocol stack is composed of four hierarchical layers. The top Function Layer defines all supported DIMM-Link functions, such as remote memory access, synchronization, CPU-forwarding request (see Section IV-A), etc. These functions are implemented via three underlying layers: a Transaction Layer, a Data Link Layer, and a Physical Layer. Details are described as follows.

Function Layer. The function layer defines a set of high-level functions supported via DIMM-Link, which are called *DL functions*. Here we briefly introduce two basic functions: *(1) Memory Access:* When the NMP cores need to read or write remote DRAMs, the inter-DIMM memory access requests will be issued and transmitted to the destination DIMMs automatically. Apart from point-to-point read/write, DIMM-Link also supports broadcasting packets, which demands explicit API calls in programs, to improve communication efficiency in broadcast-dominant applications [76]. *(2) Synchronization:* Parallel-computing applications widely use synchronization primitives [12], [19], [20], [42], [43] to synchronize the processing cores, and NMP workloads are no exception. Previous HMC-based NMP accelerators adopt barriers [1], [24], [88], [91] and locks [59] to synchronize among

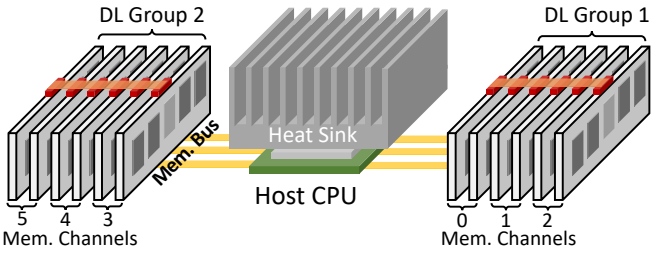


Fig. 4. An Illustration of DIMM-Link Groups.

TABLE II
SERDES TECHNIQUES COMPARISON

Reference	[10]	[25]	[69] (GRS)
Media	SMA Cable	Ribbon Cable	PCB
Singal Rate	6Gb/s/pin	16Gb/s/pin	25Gb/s/pin
Reach	953mm	500mm	80mm
Energy Eff. (pJ/b)	0.58	2.58	1.17

NMP cores. The host CPU may also need to synchronize with the NMP cores if it participates in the computation [9], [76]. However, for existing DIMM-NMP architectures, synchronization can only be implemented via inefficient host CPU polling [32], [76] or by replicating the finite-state machines (FSMs) of NMP accelerators and placing them in the host-side controller [9]. DIMM-Link supports message-passing-based synchronization and minimizes data traffic through hierarchical synchronization (See Section III-D).

Transaction Layer. The transaction layer turns the DL functions' data into packets, whose format is defined in Figure 3-(b). Each packet has a header and a tail, between which are the optional data payloads. The whole packet is sliced into one or more 128-bit flits. The 64-bit header contains an `SRC` and a `DST` field to specify the unique IDs of both source and destination DIMMs. For broadcast packets, the `DST` field will be ignored, and any DIMMs can accept the packet. The following 4-bit `CMD` field carries the command of this transaction, which is given by the DL functions. The `ADDR` information is required for address-based operations such as memory access. We use 42 bits to address up to 4TB of memory. Considering that the destination ID bits have already been used in the address mapping, we only store the remaining 37 bits in the `ADDR` field for efficiency. A `TAG` field uniquely identifies the request and response (optional) packets of a transaction. Finally, we use a 5-bit `LEN` field to record the number of flits in this packet. `LEN=0` means there is only one flit (e.g., packets for memory read requests have no data payload). Each packet contains up to 32 flits, carrying 256 bytes of payload data.

Data Link Layer. The data link layer (DLL) provides reliable transport of transaction packets from one DIMM to another. As Figure 3-(b) shows, there is a CRC (Cyclic Redundancy Check) field and a DLL field in the tail of each packet. When the packet arrives at the destination DIMM, the CRC checker in the router will detect data errors in the packet by validating the 32-bit CRC. Then, the DLL field is used for retry control. An ACK packet is sent back to the source if the CRC is correct.

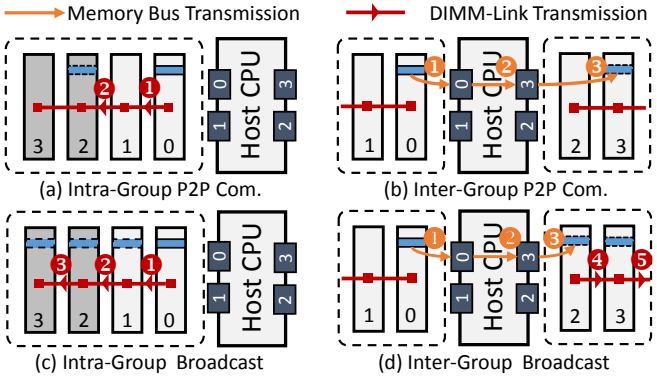


Fig. 5. Four Inter-DIMM Communication Patterns.

The packets should be re-transmitted if the source DIMM does not receive the ACK within a specified time. The DLL field also contains credit bits to support flow control.

Physical Layer. The physical layer in Figure 3 contains digital and analog circuits to support the serialization, transmission, and deserialization of DL packets, namely the DL-PHY component and the DL-Bridge in Figure 2. More details about the physical layer are introduced in Section III-D.

C. DIMM-Link Group and Hybrid Routing

DIMM-Link Group. As Figure 4 shows, DIMMs are usually distributed on both sides of the host CPU on the motherboard of commodity servers [81]. While all DIMMs on the same side of the CPU are connected by a DL-Bridge to form a "DIMM-Link (DL) Group", the DIMMs on different sides are not directly connected. The reasons for this design are explained as follows. First, the CPU heat sink makes it difficult to connect DIMMs from two sides with a DL-Bridge directly. Second, the distance between DIMMs on both sides is too long to use high-bandwidth off-package serial links like GRS [69], which has limited reach (about 80 mm). As compared in Table II, though we can alternatively use some long-reach SerDes techniques, the bandwidth and energy efficiency are usually not comparable with GRS. Also, the SMA/Ribbon-cable-based links have a much lower integration density than the PCB-based GRS link [16], substantially increasing the difficulty of hardware design.

Hybrid Routing. Under Figure 4's organization, packets are transmitted via DIMM-Link directly if DIMMs are in the same group. For inter-group packets, they are still forwarded via the host CPU. Therefore, a hybrid routing mechanism is required. As illustrated in Figure 5, there are four different IDC types: *(a):Intra-Group P2P Communication.* When both the source and destination DIMMs are in the same group, the packets will be routed via DIMM-Link only. For example, if DIMM-0 in Figure 5-(a) writes data to DIMM-2, the packets undergo a two-hop routing (Steps ❶-❷).

(b):Inter-Group P2P Communication. For point-to-point communication, if the source and destination are in different groups, the packets have to be forwarded by the host CPU. In Figure 5-(b), if DIMM-0 wants to write data to DIMM-3, the host CPU first gets the transmission requests via periodical

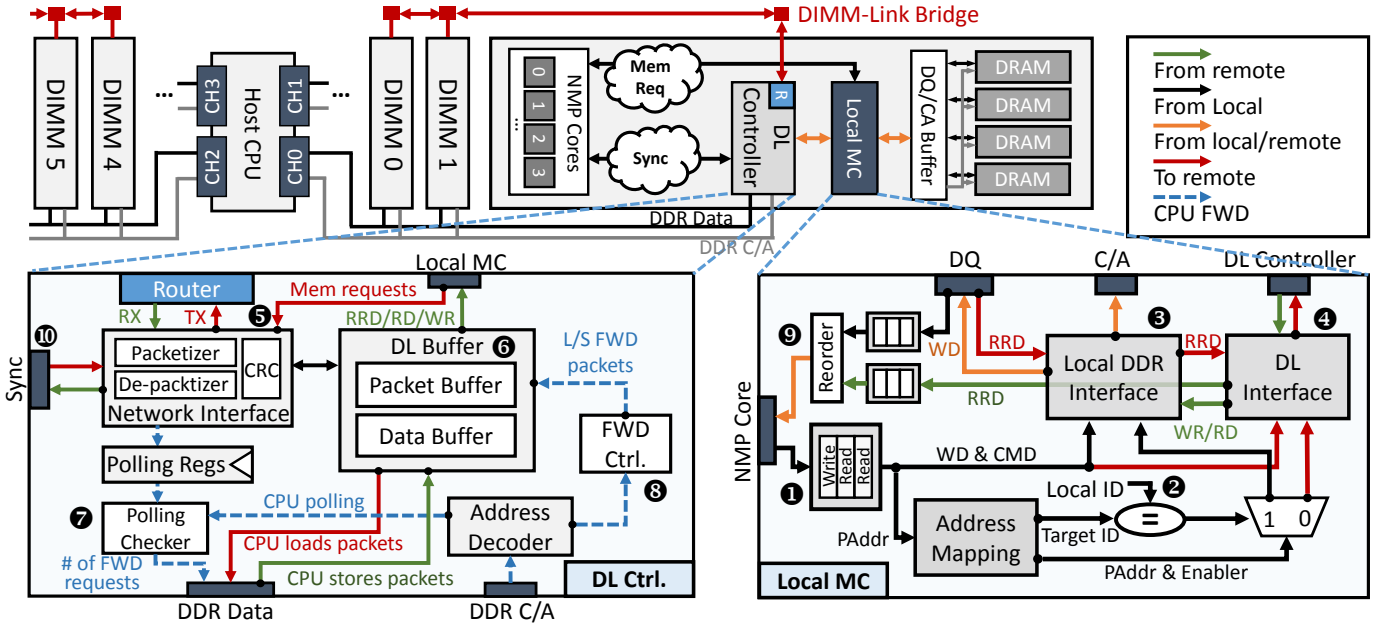


Fig. 6. Detailed Architecture Design of DIMM-Link.

polling and then reads the data packet from DIMM-0 in channel-0 (Step ❶). The packet is forwarded to channel-3 at Step ❷ and finally written to DIMM-3 at Step ❸.

(c):Intra-Group Broadcast. For inter-DIMM broadcast within the same group, the packets are routed via DIMM-Link until all the destination DIMMs have received the packet. For example, broadcast data from DIMM-0 to the remaining DIMMs in Figure 5-(c) requires three routing steps.

(d):Inter-Group Broadcast. An example of the inter-group broadcast is shown in Figure 5-(d). It is completed in two phases, which include inter-group P2P access (Steps ❶-❷-❸) and intra-group broadcast (Steps ❹-❺). Both the host-CPU forwarding and DIMM-Link transmission are involved.

Three issues should be addressed to realize an efficient hybrid routing mechanism. First, the DL-Controller should support data transmissions via both DIMM-Link and the memory bus. Second, for inter-group IDC, the host CPU forwarding still suffers from low bandwidth and long latency. Periodical CPU polling is also inevitable. Third, in the same group, the IDC latency increases with the routing hops. To enable and optimize the hybrid routing, we carefully design the architectures of the DL-Controller and Local Memory Controller in Section III-D. We also propose two optimization strategies to mitigate the CPU-forwarding and DIMM-Link transmission overhead in Section IV.

D. Detailed Architecture Design

In this section, we dive into more detailed architecture designs of DIMM-Link to show how the DL functions and the hybrid routing mechanism are efficiently implemented. The introduction is expanded around the support for two basic DL functions: Memory Access and Inter-DIMM Synchronization.

Support for Memory Access. We customize the Local Memory Controller (Local MC) and the DL-Controller to implement the hybrid routing and make memory access application-transparent. As Figure 6 illustrates, an NMP core's memory access request is first sent to the Local MC, which stores these requests into a Transaction Buffer (❶). One request is read out each time, whose address is decoded by the Address Mapping module to get the target DIMM ID. An arbiter (❷) checks whether it is a local request. If so, the address, command (`Read` or `Write`) and write data (WD) are sent to the Local DDR Interface (❸) to access the local DRAM. Otherwise, they are sent to the DL-Interface (❹), which feeds the memory access requests to the DL-Controller. The Network Interface (❺) in the DL-Controller packetizes the remote memory access request and checks the DL group of the destination DIMM to choose the routing mode. According to the hybrid routing mechanism, there are two conditions:

(a) Intra-Group Transmission: If the source and destination DIMMs are in the same group, the Network Interface will feed packets into the router to send them out through the DL-Bridge. Once a packet arrives at the destination DIMM, its Network Interface will check data errors, decode the packet, and then put the memory access request into the Data Buffer (❻). The Local MC will be notified to read out the received requests through the DL-Interface (❹). If the request is a `Read` request (RD), then the Local DDR Interface (❸) will access the DRAM and send the read-return-data (RRD) to the source DIMM via the DL-Interface. If the request is a `Write` (WR) request, the data will be written to the local DRAM directly.

(b) Inter-Group Transmission: If the source and destination DIMMs are in different groups, the packets must be forwarded via the host. To this end, the Network Interface (❺) first increases a counter recorded in the Polling Regs, which indicates the number of waiting requests. The host CPU gets notified of these requests by periodical polling. Specifically, the CPU

keeps reading a special address via the DDR interface. An Address Decoder in the DL-Controller identifies such polling signals and invokes the Polling Checker (❼), which returns the number of waiting requests to the host. If any forwarding (FWD) requests exist, the host will (1) fetch the packets from the Packet Buffer (❻). (2) Decode the destination DIMM ID from these packets. (3) Store them to the Packet Buffer of the destination DIMM (Also through reading/writing special memory addresses). These operations are assisted by the FWD Controller (❽). Once the packets arrive at the destination DIMM, they will be decoded and invoke the following operations, just like an intra-group transmission. If the request is a remote read, the RRD will be packetized and forwarded back to the requesting DIMM. The concurrent RRD from both local and remote DRAMs should be reordered (❾) before being fed back to the NMP cores.

Support for Synchronization. As stated in Section III-B, synchronization is usually necessary for DIMM-NMP architectures. Therefore, DIMM-Link supports message-passing based synchronization and implements a hierarchical synchronization [26], [31], [79] to reduce the communication traffic. Specifically, in each DIMM, one of the NMP cores serves as the master core, which coordinates the local synchronization. In each DL group, we set a master DIMM. The master cores in each DIMM send aggregated messages to the master DIMM via DIMM-Link (❿ in Figure 6) to synchronize a DL group. To reduce average hops, we heuristically select the DIMM at the middle of each group as the master. Finally, the master DIMMs in different DL groups coordinate with each other to realize global synchronization. Compared to the centralized synchronization methods, such a hierarchical synchronization significantly reduces the total traffic. In addition, since the host only needs to check the master DIMMs to synchronize the host CPU and NMP-cores rather than scan all the DIMMs [32], [76], the CPU-polling overhead is significantly mitigated.

E. Design Considerations

Virtual Address Translation. Since DIMM-Link uses physical addresses to read/write data, the NMP cores' memory management strategies will not affect the design of DIMM-Link. For simplicity, like many previous NMP architectures [1], [21], [32], [44], we assume simple memory segmentation without paging for the region of memory address space accessed by NMP cores, which does not require complicated MMUs.

Data Consistency & Cache Coherence. As introduced in Section II-A, a coarse-grained execution flow requires the DIMM to go between two modes: Host Access (HA) mode and NMP Access (NA) mode. Before and after the kernels' execution, the DIMMs are in HA mode. Host CPU directly accesses DRAM data. During the execution of kernels, the DIMMs are in the NA mode. The DIMM-side memory controllers handle both the local memory access and inter-DIMM access. Although the host CPU needs to send DDR commands to DIMMs for polling and packet forwarding, it only touches data in the SRAM DL buffer and registers and thus does not actually involve memory operations. Therefore, the host and

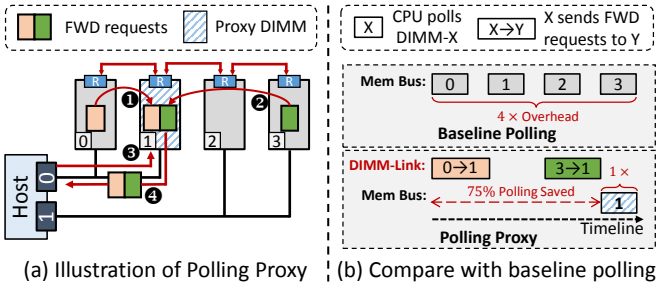


Fig. 7. The Polling Proxy Mechanism.

local MCs do not conflict with each other. Considering the caches in the host CPU and NMP cores, maintaining data coherency is also crucial. We adopt a software-assisted cache coherence [27], [31], [82] to guarantee coherence among NMP cores. Data can be either thread-private, shared read-only, or shared read-write. Thread-private and shared read-only data can be cached by NMP cores, while shared read-write data is uncacheable. Therefore, DIMM-Link does not need to care about the NMP caches since all the shareable data in DRAM is always up to date. To maintain the consistency between the host and NMP cores, the NMP data region is marked uncacheable to the host CPU, which is a common practice in previous works [1], [21], [32], [44]. When the kernels finish execution, the NMP cores also flush their caches so that the host CPU can fetch the results from DRAMs.

IV. Optimizations

A. The Polling Proxy Mechanism

Since the inter-group transmission still demands CPU-forwarding, the host CPU has to keep polling all the DIMMs, which occupies scarce host CPU and memory bus resources even without any forwarding requests [39]. To mitigate this problem, we propose a polling proxy mechanism. Specifically, we select a DIMM in each group as the polling proxy. Other DIMMs in the group send their forwarding requests to the proxy DIMM via DIMM-Link. In this way, the host CPU only needs to poll the proxy DIMM of each group to get the requests. The synchronization master DIMM is recommended as the polling proxy to reduce data traffic. As the example in Figure 7-(a) shows, when both DIMM-0 and DIMM-3 need CPU-forwarding, they register the requests in the proxy (DIMM-1) via DIMM-Link (Steps ❶ and ❷). The host polls DIMM-1 (Step ❸) and reads all the requests (Step ❹). After that, the host will fetch the packets in DIMM-0 and DIMM-3, respectively. Figure 7-(b) compares the baseline polling with our polling proxy mechanism. After using the polling proxy, the polling overhead is reduced by 75% in the four-DIMM case. Here we assume the CPU uses a single thread for polling. In practice, the CPU can poll multiple memory channels in parallel, which however occupies more resources.

Prior works also proposed interrupt-based on-demand polling mechanisms [3], [39]. Specifically, the DIMMs leverage an interrupt-like signal `ALERT_N` in the DDR4 standard to directly notify the host CPU of their forwarding

TABLE III
COMPARISONS OF POLLING MECHANISMS

Methods	On-demand Polling?	CPU-Polling Range	Overhead	Latency
Baseline polling	✗	All DIMMs		
Baseline polling + Interrupt	✔	DIMMs in all interrupting channels		
Polling Proxy	✗	One DIMM per group		
Polling Proxy + Interrupt	✔	One DIMM per interrupting group		

requests. However, the interrupt-based methods may incur frequent context switching, which introduces extra latency. Moreover, since the DIMMs in a channel share the same `ALERT_N` signal, the host still has to scan all the DIMMs in the channel to find the requester DIMM. Fortunately, if we combine the interrupt-based polling with the polling proxy, the host only needs to check the proxy DIMM in each group once interrupted. We compare different polling mechanisms in Table III. Theoretically, combining the polling proxy with interrupt-based polling brings the lowest overhead (measured by the memory bus occupation). A quantitative analysis of different polling mechanisms is presented in Section V-D.

B. Distance-Aware Task Mapping

To harness NMP architectures' high parallelism, previous works all adopt multi-threading/processing models [1], [3], [32], [38], [66], [76], [91]. Ideally, a thread is co-located with its frequently-accessed data to alleviate remote data access. This is easy to achieve for some domain-specific NMP accelerators [39], [44], [91] with fixed dataflow. TOM [38] also proposes an automatic offloading/mapping mechanism to improve the code-data affinity. However, they do not consider the "distance" between memories fully. For DIMM-Link, a distance-aware task mapping strategy is necessary.

Distance-Aware Thread Placement. Figure 8 illustrates the proposed distance-aware thread placement strategy. It is based on a general observation that the computing phases in multi-threading programs usually exhibit repeatable memory access patterns [38], [60], [78]. Therefore, we can approximate the optimal thread placement by analyzing a small fraction of the memory traces. Specifically, we first randomly place $\mathbf{T}$ threads to $\mathbf{N}$ DIMMs. Then the system enters a profiling phase. Each DIMM runs the assigned threads and records the amount of traffic to any DIMMs. The profiling period terminates after pre-defined cycles (e.g., 1% of total cycles). Then, the host CPU reads out and accumulates all the counters to generate a table $\mathbf{M}[\mathbf{T}][\mathbf{N}]$, where $\mathbf{M}[i][j]$ denotes the total memory access of thread i to DIMM j.

We propose an algorithm to derive the optimized thread placement using the profiling results. The process is described in Algorithm 1, which has three steps:

Step 1. For each thread i we estimate the cost of placing it to any DIMM j. A cost table $\mathbf{C}[i][j]$ records the total memory access counts weighted by the distance between DIMM j and the other DIMMs (measured by a distance function $dist(j,k)$). The heatmap (❶ in Figure 8) gives an example of the cost table

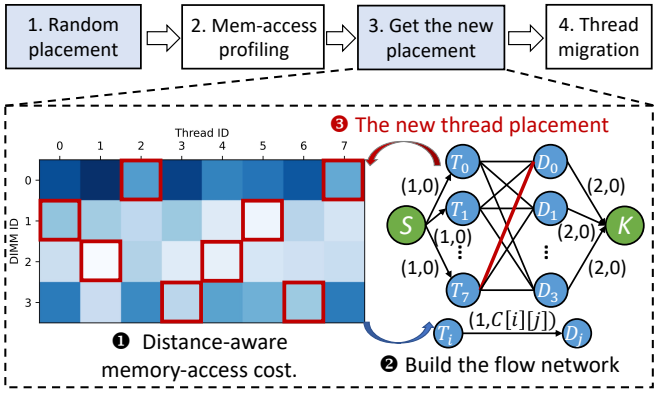


Fig. 8. Distance-Aware Thread Placement.

Algorithm 1: Distance-Aware Thread Placement

Input: Number of Threads $\mathbf{T}$; Number of DIMMs $\mathbf{N}$; Maximum threads per DIMM $\mathbf{L}$; Profiled memory access table $\mathbf{M}$;
Step 1: Estimate the distance-aware memory access cost:
▷ Initialize a cost table $\mathbf{C}[\mathbf{T}][\mathbf{N}] = \{0\}$;
for $i \leftarrow 0$ *until* $\mathbf{T}$ **do** // Traverse all the threads
  for $j \leftarrow 0$ *until* $\mathbf{N}$ **do** // Try to place thread i to DIMM j
    for $k \leftarrow 0$ *until* $\mathbf{N}$ **do** // The cost of accessing DIMM k
      $\mathbf{C}[i][j] = \mathbf{C}[i][j] + dist(j,k) * \mathbf{M}[i][k]$; //Accum. cost
Step 2: Solve a minimum-cost maximum-flow problem:
▷ Create the flow network G:
  1. Introduce a `Source` vertex and a `Sink` vertex;
  2. View each thread as a vertex. Each DIMM is also a vertex;
  3. Connect `Source` to each `Thread`. Each edge has capacity 1;
  4. Connect `Thread` i to `DIMM` j, $i \in \{0, ...\mathbf{T}-1\}$, $j \in \{0, ..., \mathbf{N}\text{-}1\}$. Edge $e(i,j)$ has capacity 1 and cost $\mathbf{C}[i][j]$;
  5. Connect `DIMM` to `Sink`. Each edge has capacity $\mathbf{L}$, cost 0.
▷ Find the minimum-cost maximum-flow of graph G using algorithms such as Bellman-Ford;
Step 3: For each flowed edge $e(i,j)$, place thread i to DIMM j.

$\mathbf{C}[\mathbf{T}][\mathbf{N}]$. Since each thread can be placed in any DIMM, it is costly to derive the optimal placement (with minimal total cost) using brute-force search algorithms. Instead, we can convert this problem to a classic minimum-cost maximum-flow problem [23], [35], [64] and solve it in polynomial time.

Step 2. As illustrated in Figure 8 (❷), we build a flow network based on the cost table. We view each thread and DIMM as a vertex. All `Thread` vertexes (T) are connected to a `Source` vertex (S). Each edge has capacity 1 and cost 0 (denoted as (1, 0)). All `DIMM` vertexes (D) are connected to a `Sink` vertex (K). Each edge has a capacity $\mathbf{L}$, which denotes the maximum threads per DIMM. In this example, $\mathbf{L}$ is 2. And the cost is also set to 0. The `Thread` and `DIMM` vertexes are connected as a bipartite graph. The edge between thread i and DIMM j has capacity 1 and cost $\mathbf{C}[i][j]$. With the flow network G, we can calculate the minimum-cost maximum-flow using algorithms like Bellman-Ford [23]. The time complexity is merely $O(\mathbf{T}^2\mathbf{N}^2)$, where $\mathbf{T}$ and $\mathbf{N}$ are the numbers of thread and DIMMs, respectively. It only takes about 2 ms on an AMD 5950X CPU to place 64 threads on 16 DIMMs.

Step 3. After calculating the maximum flow, the selected edges of the bipartite graph indicate the optimized thread placement. For example, if the edge connecting T_7 and D_0 is selected (the red edge), then Thread-7 should be placed to DIMM-0.

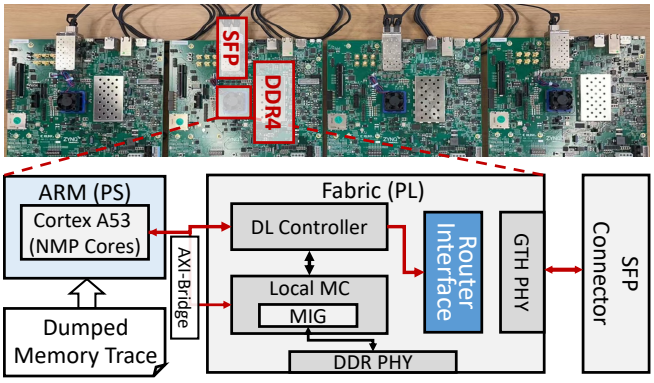


Fig. 9. The FPGA Prototype.

Finally, the threads are migrated to target DIMMs according to the derived thread placement. The migration overhead is negligible since the same program binaries have already been loaded into the NMP cores, and we only have to modify the input arguments to let them compute on new data and then restart the computing. The DIMM-side and host-side runtimes cooperate to handle such a migration.

V. Evaluation

To comprehensively evaluate DIMM-Link, we first demonstrate an FPGA-based prototype to validate its function. Then, we evaluate DIMM-Link's performance against three IDC baselines through simulation.

A. Conceptional FPGA Prototype

As presented in Figure 9, we implement a DIMM-Link prototype using multiple Xilinx ZCU-102 FPGAs. The PS-side ARM processors serve as the NMP cores. At the PL-side (Progarmmable Logic), we implement the DL-Controller and Local MC (based on the Xilinx MIG IP). The PL's DDR4 memory serves as the local memory, which can be accessed via the Local MC. We implement the network interface and a router to transmit DL packets through the GTH transceiver (adjacent FPGAs are connected via the SFP connectors). We use pre-dumped traces to drive the system. The ARM processor translates the memory traces to `Read/Write` requests and feeds them to the PL side. Due to the lack of a host CPU in such a system, we only validate intra-group operations.

Resource Utilization. We implement the hardware logic using Vitis HLS and Verilog HDL and synthesize the design with Vivado 2021.2. On each ZCU-102 FPGA board, our design only consumes 5.51% LUT, 6.23% FF, and 2.74% BRAM resources. Since the buffer chip of a DIMM can have 100 mm^2 of area [61], we believe that adding the customized logic in buffer chips only incurs negligible area (and power) overhead.

Performance & Analysis. Running at 100MHz, the FPGA prototype takes about 1.2 µs to packetize a memory write request. The CRC module contributes to most of the latency due to the inefficient implementation using the HLS-based IP [67]. Without CRC, the packet generation/decoding can finish in 18 cycles. Such a prototype partially validates the feasibility of DIMM-Link's architectures. The performance will be significantly improved if DIMM-Link is implemented in ASIC and adopts the GRS [69] link. In the next section, we perform a simulation-based evaluation to facilitate the comparison with other IDC methods.

TABLE IV
Benchmarking Applications

Name	Abbr.	Name	Abbr.
Breadth-First Search	BFS	Needleman-Wunsch	NW
Hotspot	HS	PageRank	PR
K-Means	KM	Single Source Shortest Path	SSSP

TABLE V
System Parameters and Configurations

Host CPU / CPU Baseline	
Processor	16-core @ 3.2GHz
L1I / L1D / L2 / L3	32KB,4 / 32KB,8 / 256KB,16 / 8MB,16
NMP Cores	
Processor	4-core @ 2.0GHz
L1I / L1D / L2	16KB,4 / 16KB,4 / 128KB,16
DIMM-Link Parameters	
25Gb/s/Lane, 1.17 pJ/b, 8 × Lanes (25GB/s per Link)	
Memory System	
DDR4-2400, 4Gb x72, 2 ranks/DIMM, FR-FCFS Address Mapping: `Row-Bankgroup-Bank-DIMM-Column`	
DRAM Timing Parameters	
tRC=56, tRCD=17, tCL=17, tRP=17, tBL=4 tCCD_S=4, tCCD_L=6,tRRD_S=4, tRRD_L=6, tFAW=26	

B. Evaluation Methodology

Baselines. We compare DIMM-Link with three baseline IDC methods: MCN [3] (similar to UPMEM [32]), which relies on CPU forwarding. AIM [11], which adopts a dedicated memory bus and ABC-DIMM [76] which broadcasts data within each memory channel. MCN and AIM are mainly used for comparing the point-to-point communication performance, while ABC-DIMM is used to compare the broadcast performance.

Benchmarks. Table IV lists the benchmarking tasks. Following previous NMP works [5], [21], [32], [87], we implement four typical memory-intensive tasks: Breadth-First Search (BFS), Hotspot (HS), K-Means (KM), and Needleman-Wunsch (NW). We also implement the Pagerank (PR) and Single Source Shortest Path (SSSP) tasks used in ABC-DIMM [76], with the Livejournal [74] graph as the input. We choose these workloads not only because they have been widely used for evaluating previous NMP architectures, but also because they all involve frequent IDC operations, which can fully reveal the IDC performance of DIMM-Link and the baselines. All programs are parallelized using OpenMP.

Simulation. We modify the MultiPIM [86] simulator developed on Zsim [71], Ramulator [50], and BookSim [41], to simulate DIMM-Link and the baseline architectures. Considering that MultiPIM cannot directly simulate the host CPU polling and forwarding, we also view the host CPU as a routing node that takes certain cycles to forward a packet. The fixed CPU forwarding latency is profiled additionally via GEM5 [6].

Configuration. Table V summarizes the system configurations. We set a 16-core OoO CPU as the host CPU baseline.

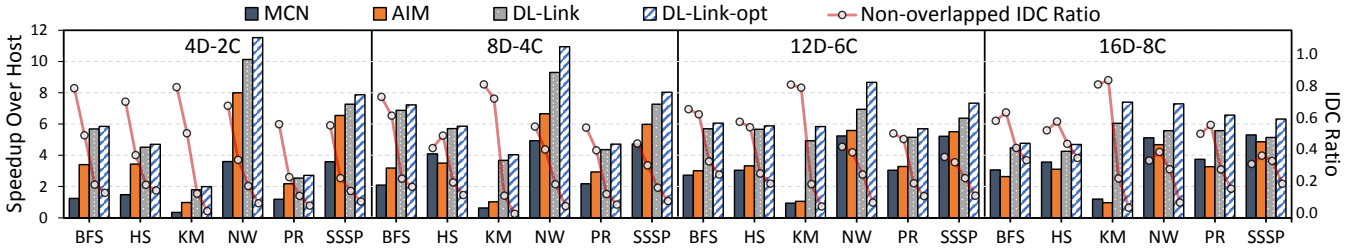


Fig. 10. Performance and Non-overlapped IDC Ratio Comparisons.

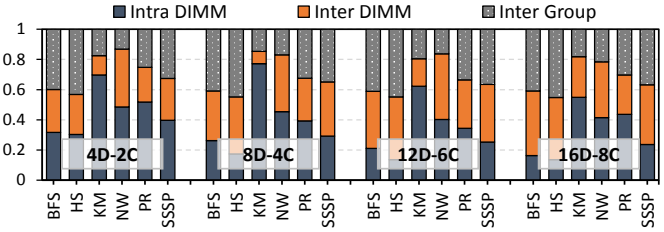


Fig. 11. Data Transfer Breakdown of DL-Link-opt.

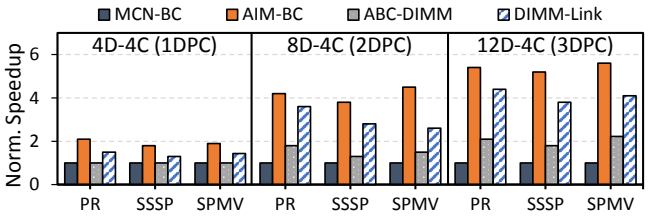


Fig. 12. Broadcast Performance Comparison.

For DIMM-NMP systems, we equip four general-purpose cores in each DIMM, which share a 128 KB L2 cache. Similar to prior works [44], [68], the NMP cores can access local ranks in parallel. Thus, the aggregated memory bandwidth is proportional to the total number of ranks. We set the DRAM timing parameters according to Micron's LR-DIMM datasheet [62]. We adopt the parameters of GRS [69] to configure the DIMM-Link interconnects. The default bandwidth is 25GB/s per bidirectional link. We profile the first 1% of total memory access for the distance-aware task mapping (similar to TOM's configurations [38]). The distance function *dist(j,k)* in Algorithm 1 is derived from profiling the latency between each pair of DIMMs. We ignore the negligible thread migration overhead, because we simply restart the computing (with new thread indices) after migration rather than checkpoint-restore all the temporal data. For the dedicated bus of AIM, we assume it has the same bandwidth as the memory bus.

C. Performance and Energy Consumption

P2P IDC Performance. We first compare DIMM-Link's performance against MCN and AIM on P2P IDC tasks. We set four system configurations, namely 4D-2C (denotes four DIMMs and two channels), 8D-4C, 12D-6C, and 16D-8C. Except for the 4D-2C configuration, which has one DL group, the other configurations have two DL groups. For all configurations, we ensure that each DIMM runs four threads. We adopt the polling proxy strategy (Table III) on DIMM-Link and use the baseline polling for MCN. We set DIMM-Link configurations both with (`DIMM-Link-opt`) and without (`DIMM-Link-base`) the distance-aware task mapping optimization. As Figure 10 shows, we plot the speedup over the 16-Core host CPU as bar graphs and use line graphs to illustrate the ratio of non-overlapped IDC cycles. `DIMM-Link-opt` achieves 5.93× geomean speedup over the CPU baseline (including the profiling time, which takes up 2% to 9%, according to the number of DIMMs). Compared to MCN, AIM and `DIMM-Link-base`, it also achieves 2.42×, 1.87× and 1.12× higher geo-mean performance, respectively.

Among all the tasks, PR and KM show strong scaling with the help of DIMM-Link. NW achieves the highest performance with four DIMMs, while BFS, HS, and SSSP prefer the 8D-4C configuration. On the one hand, the memory access patterns of different tasks affect their scalability. On the other hand, we infer that as the number of DIMMs in each group increases, the diameter (denotes the maximum hops between two DIMMs) of the DIMM-Link network also becomes larger, which may degrade DIMM-Link's performance due to the higher latency and congestion rate. We discuss this problem in Section VI. DIMM-Link reduces 55.8% of the non-overlapped IDC cycles compared to the MCN baseline, which is the key source of performance improvement. With the thread placement optimization, `DIMM-Link-opt` further reduces 48.5% (4D-2C) to 53.6% (16D-8C) of non-overlapped IDC cycles and contributes up to 1.21× end-to-end speedup (16D-8C) over the `DIMM-Link-base` systems.

The performance of MCN improves as more memory channels are equipped, which is in line with the statement [3]. However, eight channels almost reach the physical limitations due to the CPU pin count constraint. DIMM-Link's packet routing is more efficient than MCN's CPU-forwarding. Although DIMM-Link also demands CPU-forwarding for inter-group IDC, with the thread-placement optimization, only 29% of total traffic has to be forwarded via CPU according to our profiling (The data transfer breakdown of `DIMM-Link-opt` is shown in Fig. 11). In addition, we find that AIM outperforms MCN in most cases due to the avoidance of CPU-forwarding. However, AIM's performance decreases as more DIMMs are connected due to its unscalable IDC bandwidth.

Broadcast Performance. To evaluate the broadcast performance, we implement three typical applications (PR, SSSP and SPMV) in a broadcast manner following ABC-DIMM's settings. Besides ABC-DIMM, we also set MCN-BC and AIM-BC, which enable broadcast in MCN and AIM as the baselines. As Figure 12 shows, for the practical 2DPC (DIMMs-per-Channel) and 3DPC systems, ABC-DIMM's speedup over MCN-BC is not significant. This is in line with their re-

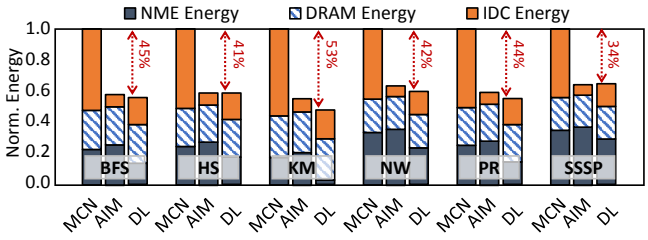


Fig. 13. Energy Consumption Breakdown.

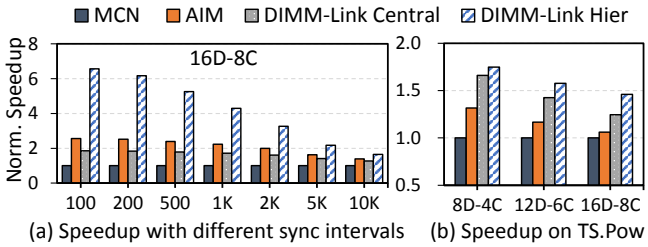


Fig. 14. Synchronization Performance.

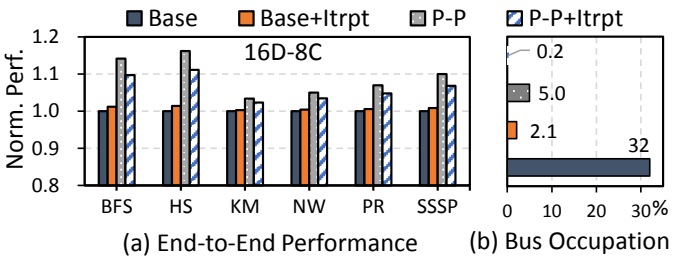


Fig. 15. Performance with Different Polling Methods.

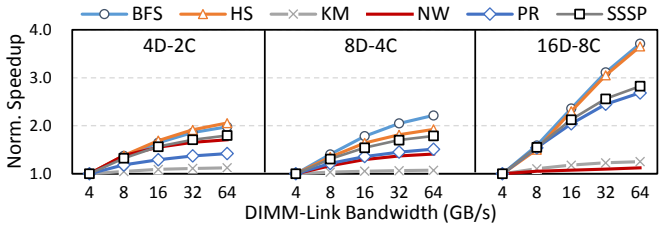


Fig. 16. DIMM-Link Bandwidth Exploration

sults [76]. Compared to MCN-BC and ABC-DIMM, DIMM-Link achieves 2.58× and 1.77× geomean speedup, respectively. DIMM-Link breaks the DPC limitation of ABC-DIMM by connecting more DIMMs together, which greatly improves broadcast efficiency and mitigates the overhead of CPU-forwarding. We can see that AIM-BC outperforms DIMM-Link under all settings as we assume AIM can broadcast data to all DIMMs via the dedicated bus, without any routing or CPU forwarding. However, as pointed out before, connecting many DIMMs with a multi-drop bus is impractical for modern DDR4/DDR5 DIMMs. Our DIMM-Link architecture perfectly balances the performance and implementation feasibility.

Energy Efficiency. Figure 13 compares the energy consumption of different IDC methods. The power of DIMM-Link is set to 1.17pJ/b according to GRS [69]. We follow a prior work [44] and set the DDR `Activate` energy to 2.1 nJ. Each DDR `RD/WR` operation consumes 14pJ/b. The off-chip IO via memory bus consumes 22pJ/b. The power of a four-core NMP processor is estimated as 1.8W according to MCN [3]. We estimate the CPU forwarding energy by profiling each polling and forwarding operation with GEM5 [6] and McPAT [55]. We assume that AIM consumes the same energy per bit as the memory bus [11]. As we can see, on the 16D-8C configuration, DIMM-Link reduces 1.76× energy consumption compared to MCN (on average), most of which comes from the reduced IDC energy. AIM consumes the lowest IDC energy since it directly connects all DIMMs with a bus and does not involve the energy-consuming CPU forwarding. However, such an architecture ignores the crucial signal integrity challenge and system integration constraints. Still, DIMM-Link achieves 1.07× energy saving compared to AIM, thanks to the end-to-end performance improvement presented before.

D. Sensitivity Analysis

Synchronization. The workloads in Table IV do not involve frequent synchronization. Therefore, to fully demonstrate DIMM-Link's ability to accelerate synchronization, we follow SynCron's [31] practice and evaluate the performance with variable synchronization frequencies. Except for the hierarchical synchronization with DIMM-Link (`DIMM-Link-Hier`), the other three baselines, namely MCN, AIM, and `DIMM-Link-Central`, all choose a centralized NMP core as the master. As Figure 14-(a) shows, when the synchronization interval narrows, `DIMM-Link-Hier` achieves considerable speedup over the baselines. For an interval of 500 instructions, `DIMM-Link-Hier` outperforms MCN and AIM by 5.3× and 2.2×, respectively. We also adopt the representative `TS.Pow` task used by SynCron [31] to evaluate the end-to-end performance. As shown in Figure 14-(b), `DIMM-Link-Hier` is 1.46× to 1.74× faster than MCN. Therefore, DIMM-Link can effectively accelerate synchronization-rich tasks through the flexible packet-based IDC and the hierarchical synchronization mechanism.

Polling Strategies. To prove the advantages of the proposed polling proxy strategy, we compare the end-to-end performance and memory bus occupation against the baselines listed in Table III. Evaluations are under the 16D-8C configuration. As Figure 15-(b) shows, the baseline polling (`Base`) has the highest bus occupation (32%) due to the costly per-DIMM scanning. With interruption, the bus occupation greatly reduces since the host CPU only scans the interrupting channels. Our polling proxy (`P-P`) achieves comparable low bus occupation with `Base+Itrpt` since the host CPU only polls the proxy DIMM in each group. With interruption (`P-P+Itrpt`), the memory bus occupation is further reduced to merely 0.2%.

As shown in Figure 15-(a), the `Base+Itrpt` solution has better end-to-end performance than the non-interrupt counterpart, mainly due to the reduced memory bus occupation. However, since the interrupt-based methods need to invoke the host-side interrupt-handling process to scan the channels, its forwarding latency is higher than that of the polling proxy. Therefore, the polling proxy achieves the highest end-to-end performance. Considering that the `ALERT_N` signal of DDR4 is usually used for other purposes such as parity error detection [63], using the polling proxy avoids modifying the host system to support such an interruption mechanism.

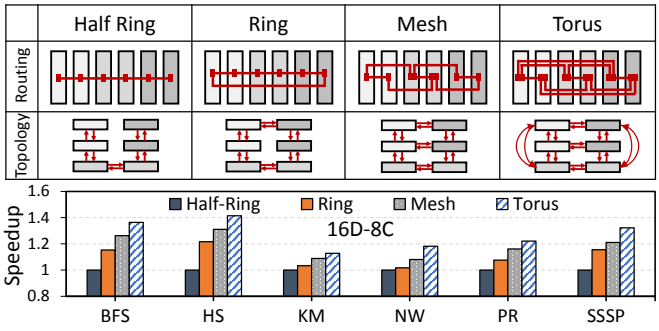


Fig. 17. Exploration of Other Topologies.

The Bandwidth of DIMM-Link. Finally, we explore DIMM-Link's performance under different bandwidth settings from 4GB/s to 64GB/s. As shown in Figure 16, from 4D-2C to 16D-8C, the benefits of increasing the bandwidth become more significant. Under the 16D-8C configuration, the performance improves almost linearly on HS and BFS tasks. This phenomenon supports our inference that a large network diameter may result in longer latency and higher congestion rate and constrain the overall performance. We consider reducing the diameter of DIMM-Link for potentially better performance, which is discussed in the next section.

VI. Discussion and Future Work

Adopting Other Topologies. Giving priority to the implementation feasibility, in this paper our DIMM-Link prototype only connects the adjacent DIMMs with bidirectional data links to form a Half-Ring topology. However, as the number of DIMMs increases, the *diameter* of DIMM-Link becomes larger, which may degrade DIMM-Link's performance due to the higher latency and congestion rate. Therefore, we also explore some other topologies listed in Figure 17. DIMMs within a DL group are connected as Ring, Mesh, or Torus. Their performances in a 16D-8C system are also shown in the figure. Compared to the baseline topology, on average, the Ring, Mesh, and Torus topologies can accelerate the P2P IDC performance by $1.11\times$, $1.19\times$ and $1.27\times$, respectively.

However, implementing the Ring topology in DIMM-Link demands a long-reach interconnect (Table II). The more complex Mesh and Torus topologies not only require the network interface in DL-Controllers to provide more ports for connection but also significantly increase the placement&routing and signal integrity challenges. The performance/overhead trade-off needs more in-depth evaluations. The linear organization is the most practical design and can already outperform the baseline methods on the evaluated workloads.

DIMM-Link on Disaggregated Memory. We argue that DIMM-Link's application is not restricted to traditional memory organizations. In recent years, disaggregated memory [7], [34], [56] has become a practical solution to break the memory capacity limitation of a single server. It organizes DIMMs as memory blades connected to the host CPU via PCIe, CXL, or Ethernet. If a memory blade equips DIMM-NMPs, DIMM-Link can be used to augment the local IDC capability. Existing protocols like RDMA or CXL can still realize inter-blade communication. However, it requires support from the software stack and programming model aspects.

Adapt DIMM-Link to Other Architectures. In this paper, we adopt the most popular centralized buffer-chip architecture to demonstrate how to integrate DIMM-Link into a DIMM-NMP architecture. For other DIMM-NMP architectures based on separate buffer chips [5] or using near-bank processing [32], the DL-Controller should be implemented as a standalone module. The distributed NMP cores in a DIMM should have data paths to the centralized DL-Controller to send and receive memory access requests.

VII. Related Work

Many near-memory processing accelerators using 3D/2.5D-stacked memories have been proposed for graph processing [1], [13], [65], [88], [91], DNN acceleration [29], [36], [47], [51], [54], [57], [72], [83], [85], or general-purpose applications [21], [26], [33], [38], [84], [87]. Other NMP architectures are DIMM-based: NDA [21] stacks the computation units atop DRAM chips of DIMMs. Chameleon [5] puts CGRA cores to the separated buffer chips of DDR4 LR-DIMMs. TensorDIMM [52], RecNMP [44], TRiM [68] and FAFNIR [4] accelerate recommendation systems with near-memory tensor reduction. ENMC [58] and StepStone [8] focus on sparse tensor algebra in deep-learning applications. Medal [39] designs a DIMM-NMP accelerator to accelerate DNA seeding. GNNear [90] proposes a hybrid architecture combing both DIMM-based NMP and centralized acceleration engine for efficient full-batch GNN training. However, DIMM-NMP architectures lack an effective IDC mechanism which limits its application in many general-purpose applications [32]. Though ABC-DIMM [76] and AIM [11] also propose to optimize IDC, they fail to fully consider implementation and system integration challenges. Our DIMM-Link presents a full-stack design to enhance the inter-DIMM communication performance for generic DIMM-NMP architectures.

VIII. Conclusion

This paper proposes the DIMM-Link interconnect to enable efficient inter-DIMM communication (IDC) in DIMM-NMP architectures. We present the full-stack design of DIMM-Link including the hardware architecture, interconnect protocol, system organization and routing mechanisms, etc. We also propose two optimization strategies to further enhance DIMM-Link's performance. Compared to three baseline IDC methods, DIMM-Link shows $2.42\times$ and $1.87\times$ higher performance than MCN and AIM, respectively. On broadcast-based tasks, DIMM-Link is also $1.77\times$ faster than ABC-DIMM.

Acknowledgment

We thank all reviewers of MICRO-2022 and HPCA-2023 for their valuable comments. We are particularly grateful to Ruifan Xu for his help about the thread-placement algorithm. This work is supported by NSF China (Grant No. 61832020, 62032001, 92064006), Beijing Academy of Artificial Intelligence (BAAI), and 111 Project (B18001).

REFERENCES

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 105–117.

[2] J. Ajanovic, "Pci express*(pcie*) 3.0 accelerator features," *Intel Corporation*, vol. 10, pp. 2–2, 2008.

[3] M. Alian, S. Min, H. Asgharimoghaddam, A. Dhar, D. K. Wang, T. Roewer, A. J. McPadden, O. O'Halloran, D. Chen, J. Xiong, D. Kim, W. W. Hwu, and N. S. Kim, "Application-transparent near-memory processing architecture with memory channel network," in *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018*, 2018, pp. 802–814. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00070

[4] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S. K. Lim, and H. Kim, "FAFNIR: accelerating sparse gathering by using efficient near-memory intelligent reduction," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*. IEEE, 2021, pp. 908–920. [Online]. Available: https://doi.org/10.1109/HPCA51647.2021.00080

[5] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[7] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 79–92. [Online]. Available: https://doi.org/10.1145/3445814.3446713

[8] B. Y. Cho, J. Jung, and M. Erez, "Accelerating bandwidth-bound deep learning inference with main-memory accelerators," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.

[9] B. Y. Cho, Y. Kwon, S. Lym, and M. Erez, "Near data acceleration with concurrent host access," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 818–831.

[10] W. Choi, G. Shu, M. Talegaonkar, Y. Liu, D. Wei, L. Benini, and P. K. Hanumolu, "3.8 A 0.45-to-0.7v 1-to-6gb/s 0.29-to-0.58pj/b source-synchronous transceiver using automatic phase calibration in 65nm CMOS," in *2015 IEEE International Solid-State Circuits Conference, ISSCC 2015, Digest of Technical Papers, San Francisco, CA, USA, February 22-26, 2015*. IEEE, 2015, pp. 1–3. [Online]. Available: https://doi.org/10.1109/ISSCC.2015.7062928

[11] J. Cong, Z. Fang, M. Gill, F. Javadi, and G. Reinman, "AIM: accelerating computational genomics through scalable and noninvasive accelerator-interposed memory," in *Proceedings of the International Symposium on Memory Systems, MEMSYS 2017, Alexandria, VA, USA, October 02 - 05, 2017*. ACM, 2017, pp. 3–14. [Online]. Available: https://doi.org/10.1145/3132402.3132406

[12] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[13] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "Graphh: A processing-in-memory architecture for large-scale graph processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640–653, 2018.

[14] G. Dai, Z. Zhu, T. Fu, C. Wei, B. Wang, X. Li, Y. Xie, H. Yang, and Y. Wang, "Dimmining: pruning-efficient and parallel graph mining on near-memory-computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 130–145.

[15] B. Dannan, "Signal integrity characterization of via stubs on high-speed ddr4 channels," https://www.signalintegrityjournal.com/articles/1731-signal-integrity-characterization-of-via-stubs-on-high-speed-ddr4-channels.

[16] A. Deutsch, "Electrical characteristics of interconnections for high-performance systems," *Proceedings of the IEEE*, vol. 86, no. 2, pp. 315–357, 1998.

[17] F. Devaux, "The true processing in memory accelerator," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 2019, pp. 1–24.

[18] A. Devic, S. B. Rai, A. Sivasubramaniam, A. Akel, S. Eilert, and J. Eno, "To pim or not for emerging general purpose processing in ddr memory systems," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 231–244.

[19] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt, "Parallel application memory scheduling," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 362–373.

[20] A. Elafrou, G. Goumas, and N. Koziris, "Conflict-free symmetric sparse matrix-vector multiplication on multicore architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–15.

[21] A. F. Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: near-dram acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*. IEEE Computer Society, 2015, pp. 283–295. [Online]. Available: https://doi.org/10.1109/HPCA.2015.7056040

[22] D. Foley and J. Danskin, "Ultra-performance pascal gpu and nvlink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[23] A. for Competitive Programming, "The minimum-cost maximum-flow algorithm," https://cp-algorithms.com/graph/min_cost_flow.html.

[24] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architectures and Compilation, PACT 2015, San Francisco, CA, USA, October 18-21, 2015*. IEEE Computer Society, 2015, pp. 113–124. [Online]. Available: https://doi.org/10.1109/PACT.2015.22

[25] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 113–124.

[26] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architectures and Compilation, PACT 2015, San Francisco, CA, USA, October 18-21, 2015*. IEEE Computer Society, 2015, pp. 113–124. [Online]. Available: https://doi.org/10.1109/PACT.2015.22

[27] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architectures and Compilation, PACT 2015, San Francisco, CA, USA, October 18-21, 2015*. IEEE Computer Society, 2015, pp. 113–124. [Online]. Available: https://doi.org/10.1109/PACT.2015.22

[28] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Y. Chen, O. Temam, and J. Carter, Eds. ACM, 2017, pp. 751–764. [Online]. Available: https://doi.org/10.1145/3037697.3037702

[29] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[30] K. Gharibdoust, A. Tajalli, and Y. Leblebici, "10.3 a 7.5mw 7.5gb/s mixed nrz/multi-tone serial-data transceiver for multi-drop memory interfaces in 40nm cmos," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015, pp. 1–3.

[31] C. Giannoula, N. Vijaykumar, N. Papadopoulou, V. Karakostas, I. Fernandez, J. Gómez-Luna, L. Orosa, N. Koziris, G. Goumas, and O. Mutlu, "Syncron: Efficient synchronization support for near-data-processing architectures," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 263–276.

[32] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture," *arXiv preprint arXiv:2105.03814*, 2021.

[33] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "ipim: Programmable in-memory image processing accelerator using near-bank architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 804–817.

[34] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 417–433. [Online]. Available: https://doi.org/10.1145/3503222.3507762

[35] M. Hadji and D. Zeghlache, "Minimum cost maximum flow algorithm for dynamic resource allocation in clouds," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 876–882.

[36] B. Hong, Y. Ro, and J. Kim, "Multi-dimensional parallel training of winograd layer on memory-centric architecture," in *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018*. IEEE Computer Society, 2018, pp. 682–695. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00061

[37] HPE, "Hpe proliant dl380 gen9 server," https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c05240460.

[38] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent offloading and mapping (tom) enabling programmer-transparent near-data processing in gpu systems," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 204–216, 2016.

[39] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, "Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 587–599.

[40] Intel, "Cxl," https://www.computeexpresslink.org.

[41] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "Booksim 2.0 user's guide," *Standford University*, p. q1, 2010.

[42] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt, "Bottleneck identification and scheduling in multithreaded applications," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 223–234, 2012.

[43] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt, "Utility-based acceleration of multithreaded applications on asymmetric cmps," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 154–165, 2013.

[44] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 790–803.

[45] L. Ke, X. Zhang, J. So, J.-G. Lee, S.-H. Kang, S. Lee, S. Han, Y. Cho, J. H. Kim, Y. Kwon *et al.*, "Near-memory processing in action: Accelerating personalized recommendation with axdimm," *IEEE Micro*, 2021.

[46] C. Kim, H.-W. Lee, and J. Song, "Memory interfaces: past, present, and future," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 23–34, 2016.

[47] D. Kim, T. Na, S. Yalamanchili, and S. Mukhopadhyay, "Deeptrain: A programmable embedded platform for training deep neural networks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2360–2370, 2018. [Online]. Available: https://doi.org/10.1109/TCAD.2018.2858358

[48] H. Kim, H. Park, T. Kim, K. Cho, E. Lee, S. Ryu, H.-J. Lee, K. Choi, and J. Lee, "Gradpim: A practical processing-in-dram architecture for gradient descent," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 249–262.

[49] J. H. Kim, S.-h. Kang, S. Lee, H. Kim, W. Song, Y. Ro, S. Lee, D. Wang, H. Shin, B. Phuah *et al.*, "Aquabolt-xl: Samsung hbm2-pim with in-memory processing for ml accelerators and beyond," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–26.

[50] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, 2016. [Online]. Available: https://doi.org/10.1109/LCA.2015.2414456

[51] Y. Kwon, S. H. Lee, J. Lee, S. Kwon, J. Ryu, J. Son, S. O, H. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "25.4 A 20nm 6gb function-in-memory dram, based on HBM2 with a 1.2tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *IEEE International Solid-State Circuits Conference, ISSCC 2021, San Francisco, CA, USA, February 13-22, 2021*. IEEE, 2021, pp. 350–352. [Online]. Available: https://doi.org/10.1109/ISSCC42613.2021.9365862

[52] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 740–753.

[53] S. Lee, J. Seo, K. Lim, J. Ko, J.-Y. Sim, H.-J. Park, and B. Kim, "A 7.8gb/s/pin 1.96pj/b compact single-ended trx and cdr with phase-difference modulation for highly reflective memory interfaces," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 272–274.

[54] Y. S. Lee and T. H. Han, "Task parallelism-aware deep neural network scheduling on multiple hybrid memory cube-based processing-in-memory," *IEEE Access*, vol. 9, pp. 68 561–68 572, 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3077294

[55] S. Li, J. H. 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 (TACO)*, vol. 10, no. 1, pp. 1–29, 2013.

[56] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 267–278. [Online]. Available: https://doi.org/10.1145/1555754.1555789

[57] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 655–668.

[58] L. Liu, J. Lin, Z. Qu, Y. Ding, and Y. Xie, "Enmc: Extreme near-memory classification via approximate screening," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1309–1322.

[59] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent data structures for near-memory computing," in *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, 2017, pp. 235–245.

[60] H. Löf and S. Holmgren, "affinity-on-next-touch: Increasing the performance of an industrial pde solver on a cc-numa system," in *Proceedings of the 19th annual international conference on Supercomputing*, 2005, pp. 387–392.

[61] P. J. Meaney, L. D. Curley, G. D. Gilda, M. R. Hodges, D. J. Buerkle, R. D. Siegl, and R. K. Dong, "The ibm z13 memory subsystem for big data," *IBM Journal of Research and Development*, vol. 59, no. 4/5, pp. 4–1, 2015.

[62] Micron, "32gb (x72, ecc, dr) 288-pin ddr4 rdimm."

[63] Micron, "Tn-40-40: Ddr4 point-to-point design guide."

[64] W. Moolman, "The maximum flow and minimum cost–maximum flow problems: Computing and applications," *Asian Journal of Probability and Statistics*, pp. 28–57, 2020.

[65] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level PIM offloading in graph computing frameworks," in *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*. IEEE Computer Society, 2017, pp. 457–468. [Online]. Available: https://doi.org/10.1109/HPCA.2017.54

[66] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi *et al.*, "A case study of {Processing-in-Memory} in {off-the-Shelf} systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 117–130.

[67] [Online], "Xilinx vitis libraries," https://github.com/Xilinx/Vitis_Libraries.

[68] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. H. Ahn, "Trim: Enhancing processor-memory interfaces with scalable tensor reduction in memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 268–281.

[69] J. W. Poulton, J. M. Wilson, W. J. Turner, B. Zimmer, X. Chen, S. S. Kudva, S. Song, S. G. Tell, N. Nedovic, W. Zhao, S. R. Sudhakaran, C. T. Gray, and W. J. Dally, "A 1.17-pj/b, 25-gb/s/pin ground-referenced single-ended serial link for off- and on-package communication using

a process- and temperature-adaptive voltage regulator," *IEEE J. Solid State Circuits*, vol. 54, no. 1, pp. 43–54, 2019. [Online]. Available: https://doi.org/10.1109/JSSC.2018.2875092

[70] Samsung, "Samsung brings in-memory processing power to wider range of applications," https://semiconductor.samsung.com/newsroom/news/samsung-brings-in-memory-processing-power-to-wider-range-of-applications/.

[71] D. Sánchez and C. Kozyrakis, "Zsim: fast and accurate microarchitectural simulation of thousand-core systems," in *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, A. Mendelson, Ed. ACM, 2013, pp. 475–486. [Online]. Available: https://doi.org/10.1145/2485922.2485963

[72] F. Schuiki, M. Schaffner, F. K. Gürkaynak, and L. Benini, "A scalable near-memory architecture for training deep neural networks on large in-memory datasets," *IEEE Trans. Computers*, vol. 68, no. 4, pp. 484–497, 2019. [Online]. Available: https://doi.org/10.1109/TC.2018.2876312

[73] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[74] Stanford, "Stanford large network dataset collection," http://snap.stanford.edu/data/index.html.

[75] J. Stuecheli, W. J. Starke, J. D. Irish, L. B. Arimilli, D. Dreps, B. Blaner, C. Wollbrink, and B. Allison, "Ibm power9 opens up a new era of acceleration enablement: Opencapi," *IBM Journal of Research and Development*, vol. 62, no. 4/5, pp. 8–1, 2018.

[76] W. Sun, Z. Li, S. Yin, S. Wei, and L. Liu, "ABC-DIMM: alleviating the bottleneck of communication in dimm-based near-memory processing with inter-dimm broadcast," in *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 2021, pp. 237–250. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00027

[77] N. Talati, H. Ye, Y. Yang, L. Belayneh, K.-Y. Chen, D. T. Blaauw, T. N. Mudge, and R. G. Dreslinski, "Ndminer: accelerating graph pattern mining using near data processing." in *ISCA*, 2022, pp. 146–159.

[78] D. Tam, R. Azimi, and M. Stumm, "Thread clustering: sharing-aware scheduling on smp-cmp-smt multiprocessors," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 47–58, 2007.

[79] X. Tang, J. Zhai, X. Qian, and W. Chen, "plock: A fast lock for architectures with explicit inter-core message passing," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 765–778.

[80] K. Therdsteerasukdi, G.-S. Byun, J. Ir, G. Reinman, J. Cong, and M. F. Chang, "The dimm tree architecture: A high bandwidth and scalable memory system," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 388–395.

[81] thomas krenn, "Optimize memory performance of intel xeon scalable systems," https://www.thomas-krenn.com/en/wiki/Optimize_memory_performance_of_Intel_Xeon_Scalable_systems.

[82] P.-A. Tsai, C. Chen, and D. Sanchez, "Adaptive scheduling for systems with asymmetric memory hierarchies," in *2018 51st Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2018, pp. 641–654.

[83] Y. Wang, W. Chen, J. Yang, and T. Li, "Towards memory-efficient allocation of cnns on processing-in-memory architecture," *IEEE Trans. Parallel Distributed Syst.*, vol. 29, no. 6, pp. 1428–1441, 2018. [Online]. Available: https://doi.org/10.1109/TPDS.2018.2791440

[84] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "Spacea: Sparse matrix vector multiplication on processing-in-memory accelerator," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*. IEEE, 2021, pp. 570–583. [Online]. Available: https://doi.org/10.1109/HPCA51647.2021.00055

[85] S. Yin, S. Tang, X. Lin, P. Ouyang, F. Tu, L. Liu, J. Zhao, C. Xu, S. Li, Y. Xie, and S. Wei, "Parana: A parallel neural architecture considering thermal problem of 3d stacked memory," *IEEE Trans. Parallel Distributed Syst.*, vol. 30, no. 1, pp. 146–160, 2019. [Online]. Available: https://doi.org/10.1109/TPDS.2018.2858230

[86] C. Yu, S. Liu, and S. M. Khan, "Multipim: A detailed and configurable multi-stack processing-in-memory simulator," *IEEE Comput. Archit. Lett.*, vol. 20, no. 1, pp. 54–57, 2021. [Online]. Available: https://doi.org/10.1109/LCA.2021.3061905

[87] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: throughput-oriented programmable processing in memory," in *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada - June 23 - 27, 2014*, B. Plale, M. Ripeanu, F. Cappello, and D. Xu, Eds. ACM, 2014, pp. 85–98. [Online]. Available: https://doi.org/10.1145/2600212.2600213

[88] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "Graphp: Reducing communication for pim-based graph processing with efficient data partition," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 544–557.

[89] Y. Zhao, C. Liu, Z. Du, Q. Guo, X. Hu, Y. Zhuang, Z. Zhang, X. Song, W. Li, X. Zhang *et al.*, "Cambricon-q: a hybrid architecture for efficient training," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 706–719.

[90] X. W. X. W. G. S. Zhe Zhou, Cong Li, "Gnnear: Accelerating full-batch training of graph neural networks with near-memory processing," in *Proceedings of the 31st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2022.

[91] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "Graphq: Scalable pim-based graph processing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 712–725.