Managing HBM Bandwidth on Multi-Die FPGAs with FPGA Overlay NoCs

Srinirdheeshwar Kuttuva Prakash
University of Waterloo, Canada
skuttuva@uwaterloo.ca

Hiren Patel
University of Waterloo, Canada
hiren.patel@uwaterloo.ca

Nachiket Kapre
University of Waterloo, Canada
nachiket@uwaterloo.ca

***Abstract*—We can improve HBM bandwidth distribution and utilization on a multi-die FPGA like Xilinx Alveo U280 by using Overlay Network-on-Chips (NoCs). The HBM in Xilinx Alveo U280 offers 8 GB of memory capacity with a theoretical maximum bandwidth of 460 GBps, but exposed all the HBM ports to the FPGA fabric in only one die. As a result, computing elements assigned to other dies must use the scarce Super Long Lines (SLLs) to access HBM bandwidth. Furthermore, HBM is fractured internally into thirty-two smaller memories called pseudo channels, connected together by a hardened and performance-limited crossbar. The crossbar enables global accesses from any of the HBM ports, but introduces several throughput bottlenecks. An Overlay Hybrid NoC combining Hoplite NoC with Butterfly Fat Trees (BFT) NoCs offers a high-performance solution for distributing HBM bandwidth across all three dies. The routing capability of the NoC can be modified to supplant the internal crossbar of Xilinx HBM for global accesses. We demonstrate this in Xilinx Alveo U280 with BFT, Hoplite, and Hybrid NoC, using synthetic benchmarks and two application-based benchmarks, Dense matrix-matrix multiplication (DMM) and Sparse Matrix-Vector multiplication (SPMV). Our experiments show that Overlay NoCs can improve the throughput by 1.26× for synthetic benchmarks and up to 1.4× for SpMV workloads.**

I. Introduction

In the post-Moore era, FPGAs are becoming essential platforms for hardware acceleration of contemporary workloads such as machine learning [1], graph analytics [2], database management [3], high-speed networking [4], among others. High Bandwidth Memory (HBM) is a game-changing technology that can provide bandwidth up to 460 GBps, five times higher than DDR4. It is an in-package DRAM memory technology [5] that uses through-silicon vias to interconnect vertical stacks of multiple DRAM chips. For example, the Xilinx Alveo U280 FPGA [6] ships with two HBM stacks totalling 8 GBs capacity with a maximum theoretical throughput of 460 GBps. The high bandwidth comes from its 32 independent channels, exposed to the fabric as 32 AXI ports, each 256 bits wide, operating at 450 MHz [7]. When the processing elements are restricted to access only the channel they are connected to, the achievable throughput is close to the theoretical maximum. However, in a realistic setting where the processing elements are allowed to access any channel, the internal crossbar of HBM fails to sustain the high throughput [8] [9] [10].

Modern FPGAs from popular vendors are built from multiple dies connected together using technologies like Xilinx Stacked Silicon Interconnect (SSI). For instance, the Alveo U280 is comprised of three such dies called Super Logic Regions (SLRs) [11] connected by a limited number of Super Long Lines (SLLs) [12]. The SLLs are also challenging to use for nets with feedbacks, such as a Valid-Ready path used in AXIS protocols. Yet, the HBM ports in Alveo U280 are accessible only from SLR0 [6]. Therefore, effectively distributing the bandwidth of HBM across all SLRs is a challenging and crucial task for a chip spanning design. This paper proposes using Overlay NoCs to overcome the aforementioned throughput bottleneck and the HBM bandwidth distribution problem. To that end, we adapt the BFT and Hoplite to support HBM interfacing and customize the routing scheme to route memory transactions to the target channel, eliminating the need for the flawed Xilinx HBM internal crossbar. Similarly, we also use NoCs to aid in SLR crossing, making it possible to distribute HBM bandwidth across all three SLRs of Xilinx Alveo U280 while also providing PE to PE communication between and within SLRs. Furthermore, We identify the limitations of the BFT and Hoplite NoC, and to address them; we propose a Hybrid NoC.

The main contributions of this paper are as follows:

1) Design and implementation of a new Hybrid NoC, which combines one dimensional Hoplite NoC with BFT NoCs. The vertical rings provide inter-SLR communication, whereas BFT NoC is used for intra-SLR communication.
2) RTL implementation of the BFT and Hoplite NoC modified for routing memory packets to their HBM AXI ports. In addition, the routing logic is also augmented to support multi-flit operations.
3) HLS-based application benchmark implementation of Dense Matrix Multiplication and Sparse Matrix-Vector multiplication. Design of a synthesizable RTL-based synthetic benchmarking PE.
4) Evaluation of the performance bottleneck on HBM throughput in Xilinx Alveo and the impact of using NoC overlays on HBM throughput utilization under various synthetic and application-based benchmarks.
5) Multi-SLR spanning implementations of the NoCs for a chip-wide bandwidth distribution with floorplans.

A repository containing our implementations of the NoCs and associated code to replicate our results is available https://git.uwaterloo.ca/watcag-public/hbm_fpga_noc.

II. Motivation & Background

A. Xilinx HBM limitations

The HBM in Alveo U280 offers a theoretical maximum throughput of 460 GBps with a memory capacity of 8 GB. This high throughput is made possible by the internal organization of the HBM [7], where the memory space is divided into two stacks. Each stack is further subdivided into 16 Pseudo Channels (PC) and controlled by eight Memory Controllers (MC). In total, the HBM is internally divided into 32 pseudo channels, each with a capacity of 256 MBs and a data width of 64 bits. The pseudo channels are exposed to the FPGA fabric as 32 256 bits AXI3 ports, and to reduce the clock frequency requirements in the fabric, the ports operate at a 4:1 frequency ratio. HBM uses 33 bits addressing, where the most significant 5 bits, 32 down to 28 points to the *home* pseudo channel and the corresponding *home* AXI port, we call these 5 bits ***port_id***.

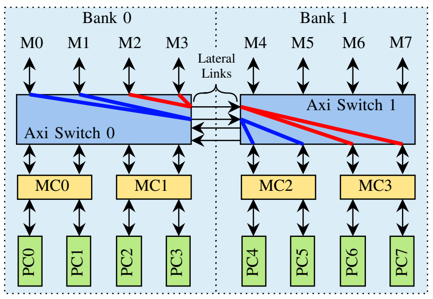


Fig. 1. A section of the internal crossbar used in HBM. Transactions from M0 to M3 that cross the bank (shown in blue and red) must share the two lateral links, limiting the throughput.

For a PE to access the entire HBM memory space, Xilinx enables access to any pseudo channel from any AXI port, using a hardened internal crossbar. In Figure 1, we show the a section of this internal crossbar. Four pseudo channels, the corresponding two memory controllers, and the four AXI ports interfaced with an AXI switch. The AXI ports from the fabric are the masters, whereas the memory controllers and the pseudo channels are the slaves [7]. We call this grouping a ***bank***. Within a bank, the AXI switch acts as a full throughput 4×4 crossbar, where any AXI port can access any pseudo channel without loss in throughput.

To support full throughput transactions between Bank 0 and Bank 1, we need four lateral links in each direction. However, only two lateral links are available. As a consequence:

1) Each lateral link is shared by two masters (Fig 1), limiting the available throughput to 50% of the maximum throughput. The throughput drops to 33% of the maximum achievable throughout for cross-stack access [13].
2) A dead cycle is required to switch between the masters during writes, which also degrades the achievable throughput, and the extent to which it degrades depends on the burst lengths [7].

B. Bandwidth Distribution across SLRs

All of the 32 HBM ports are available to interface only in SLR0 [6]. So, in a design spanning multiple SLRs, processing elements placed in SLR1 and SLR2 must use the limited and challenging to use SLLs to access SLR0 and, by extension, HBM ports. SLLs are challenging to use because they incur a considerable path delay if the SLR crossing net is not between two registers [12], lowering the attainable frequency. This is achievable through careful pipelining and floorplan constraints for strictly feed-forward design. Making this crossing can be challenging for designs with handshake protocols, such as AXI Streaming.

C. Butterfly Fat Tree (BFT)

BFT [14] [15] is a hierarchical tree-based NoC topology, where the PEs are the leaf nodes and the switches are the internal nodes of the fat tree. In this paper, we use a 2-ary BFT NoC with lightweight flow control, as described by Malik et al. [16], as our reference design.

The packets injected by PEs first ascend the network until it reaches a common ancestor between the source and the destination. In this version of the BFT NoC, the uphill freedom in the switches is limited. Packets that arrive at the L port can ascend only through U0. Similarly, packets that arrive at port R can ascend only through U1. When the packet reaches the level of the common ancestor, it turns (L$\rightarrow$ R, R$\rightarrow$ L) and starts to descend towards the destination. During the descent, no downhill freedom is available. The choice of the downhill port at level l is dictated by bit l of the destination address. If it is set, the R downhill port is chosen. Otherwise, L is chosen; This setup leads to contention between turning and descending packets. A round-robin arbitration is used to determine the winner [16]. The winner is routed to the requested port and subsequently the next level. The loser is stored in a shadow register, and the losing port gets back-pressured. The presence of shadow register and back-pressure dictates a handshake-driven protocol at each port. The AXI-streaming protocol is used as the port-level protocol for compatibility with Xilinx IPs.

D. Hoplite

Hoplite is a uni-directional 2D torus [17] [18] NoC topology where the network switches are arranged in a 2D grid-like fashion. Hoplite NoC, as proposed by Kapre et al. [19] is the reference design for the Hoplite network used in this paper. Unlike the BFT NoC, a deflection-based routing with turn restriction policy called bufferless dimension-ordered routing (DOR) is used in Hoplite. In this policy, the packets first route along the horizontal link and at the appropriate column, the packet turns and routes along the vertical link. DOR reduces the resource costs and also ensures deadlock-free operation for single flit packets. However, the in-order delivery guarantee is compromised [19].

III. Related Works

A. HBMCONNECT

HBMCONNECT, proposed by Choi et al. (2020) [9], is an HLS based FPGA interconnect designed for interfacing processing elements to the HBM. The interconnect aims to eliminate the dependence on the HBM's internal crossbar, thereby circumventing the throughput bottleneck. The interconnect is constructed using several 2x2 switching elements called Mux-Demux switch connected in a multi-level butterfly topology. Experiments show that HBMCONNECT can achieve a maximum operating frequency of 207 MHz (with Bucket Sort PE), and it improves the BW^2/LUT metric by up to 211 times and the $BW^2/BRAM$ metric by up to 85 times. However, the proposed interconnect does not support PE to PE communication and supports only PE to memory communication, and the use of full AXI protocol in their switches makes SLR crossing a challenge. Thus,

B. 2GRVI Phalanx

Jan Gray (2018) [20] has previously used a single channel Hoplite NoC architecture with a dimension 15x15 to interface 222 SC-V RV64I Processor Cluster Arrays with HBM. The 15 vertical links of the network are interfaced to 30 HBM ports via AXI-switch-MC-HBM2 bridges, with an operating frequency of 300MHz. However, this structure effectively halves the maximum theoretical throughput extracted from the HBM. It relies on the internal crossbar of HBM to provide global access, which has throughput bottlenecks.

IV. Main Idea

In this section, we present the necessary architectural and logical modifications to the BFT NoC. We also examine the advantages and the shortcomings of BFT NoC used in our designs, and based on our observations, we propose a new Hybrid architecture that aims to rectify the flaws of the BFT NoC. For the NoCs, we made the following changes:

1. PE-PE and PE-HBM communication. Processing Elements with AXIS-Compatible data paths to be interfaced with HBM ports while retaining the PE to PE communication property of the underlying NoC.

2. Homing. Memory packets are to be routed directly to the corresponding AXI port over the NoC, eliminating the traffic in the lateral links of the HBM internal crossbar.

3. Distribution of HBM bandwidth across SLRs. Modules in SLR1 and SLR2 in Alveo U280 to access HBM bandwidth by using SLL crossing through the NoC links.

A. Butterfly Fat Tree

As explained in Section II-C, BFT with lightweight flow control [16] is the starting point of our design. To interface the BFT NoC with HBM AXI ports, we use the uphill ports at the topmost network level. They are connected to the HBM ports via the HBM-NoC interface FIFOs, whereas the PEs are connected to the leaf-level ports, as shown in Figure 2. Thus, the memory requests always ascend the network, and the responses always descend the network.

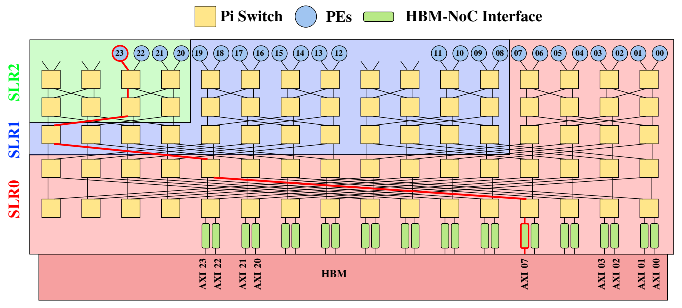


Fig. 2. PEs interfaced to HBM using BFT. Modules in the RED region are assigned to SLR0, and it also includes HBM ports. Modules in the BLUE region are assigned to SLR1, and modules in the GREEN region are assigned to SLR2. The path highlighted in red is taken by memory packets injected by PE23 with the memory address set to $70000000h$

1) Operation: In the base BFT, the upward routing freedom is limited. The choice of the uphill port is strictly based on the downhill port at which the packets arrive. While this simplifies design, it also prevents the NoC from routing the memory packets to their corresponding home AXI ports. In the BFT NoC proposed in this paper, to provide homing, the ascending memory packets can take either U0 or U1 based on the HBM memory address carried by the memory packet. At level l of the network, the uphill output port is chosen based on the bit $l+1$ of the **port_id**. Unlike other levels, at the topmost level, bit 0 determines the uphill port. If the bit is set, port U1 is chosen. Otherwise, Port U0 is chosen. No change in the routing algorithm is required for responses descending the network. In BFT, the request and responses travel in the different physical channels, and the switches use a lightweight flow control with backpressure. This ensures deadlock-free operation and in-order arrival. The original BFT network only supports single-flit routing. To support multi-flit routing for burst lengths longer than one, we incorporate a locking system in our BFT .

2) Floorplan: As per objective, we floorplan the BFT NoC with 24 PEs to span the chip across all three SLRs in the Xilinx Alveo U280 as shown in 2. Since most of the HBM subsystem modules and HBM-NoC interface FIFOs need to be assigned to SLR0, the number of NoC modules assigned to SLR0 was minimized. Considering the floorplanning restriction of Vivado and the SLL availability, 8 PEs were assigned to SLR0, 12 PEs were assigned to SLR1, and 4 PEs to SLR2. The single-flit network requires 12528 SLLs at the SLR0-SLR1 boundary and 4176 SLLs at the SLR1-SLR2 boundary, well within the available SLLs.

B. Hoplite

Hoplite NoC introduced in section III-B has been used to interface PEs to HBM. The single-channel approach used by Gray [20] is suitable for large PE counts, such as 225 PEs. However, for smaller PE counts, such as 32, the network is less flexible in terms of the usable dimensions. Instead, in our design, we use a multi-channel approach. Several Hoplite networks operate in parallel, and the processing elements can ingress and egress packets through any channel via a fan-in and

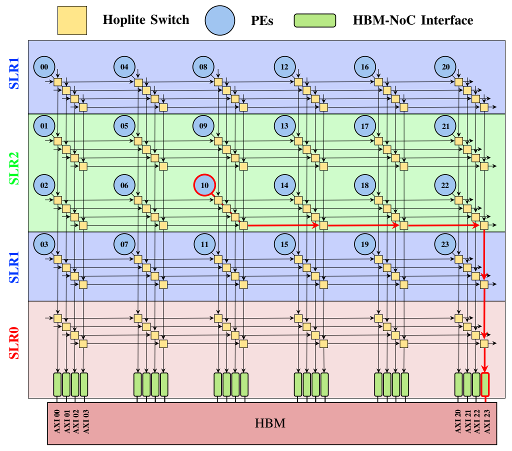


Fig. 3. PEs interfaced to HBM using Hoplite NoC. Modules in the RED region are assigned to SLR0, and it also includes HBM ports. Modules in the BLUE region are assigned to SLR1, and modules in the GREEN region are assigned to SLR2. The fan-in and fan-out modules through which the PEs ingress and egress packets are not shown. Similarly, the vertical and horizontal loops around links are not shown here for readability. The path highlighted in red is taken by memory packets injected by PE10 with memory address $170000000h$.

fan-out module, respectively. Consequently, we use a Hoplite with dimension 4×8 with four channels for our design. This ensures that all 32 ports of the HBM are used and shared by 32 PEs. With the Hoplite network arranged in this fashion, if one row of the network is floor planned to a different SLR, 19392 SLLs are required, which exceeds the availability after Vitis overheads. To remedy this, we remove eight vertical links and eight PEs, making the dimension of the network 4×6 with four channels. This change drops the SLL requirement to 14544 SLLs per SLR boundary, which is well within the available SLLs at each SLR boundary. This solution is similar to the one used by Gray [20], where to interface the network with 30 HBM ports, only 15 lateral links are used.

To interface an NoC with dimensions 4×6 (with four channels) and 24 PEs to 24 HBM ports, we use one additional row of Hoplite Switches, making the final dimension of our Hoplite NoC 5×6 (four channels), as shown in Figure 3.

1) Operation: In the base design, PEs can inject their packets into any of the available channels. However, to route the memory packets to their corresponding home HBM port, the fan-out module chooses the channel based on the memory address. The memory packets are injected into the channel determined by the equation (**port_id** $\bmod 4$).

In the Hoplite NoC, the read request and read response share the same channel, and the south port is also used as the input port to the PE. These properties of the network create a possibility of livelock for reads. We limit pending read requests to ensure livelock-free operation, and the FIFOs are sized to handle the worst-case pending reads limit.

In this paper, we do not consider Hoplite for any multi-flit experiments due to challenges of adopting the deflection based routing used in the base Hoplite design for multi-flit routing.

2) Floorplan: In Figure 3, we show a top-level floorplan used for the Hoplite NoC. The links are folded to minimize the path delay, and the folding of the vertical links can be observed from this figure. We also fold the horizontal links. However, we do not show it for improving the readability. Switches from the last row and the HBM-NoC interface FIFOs are assigned to SLR0, left unfolded. In contrast, SLR1 and SLR2 get two folded rows with 12 PEs each. This floorplanning results in 14544 SLLs at both the SLR boundaries.

C. Hybrid

We develop a Hybrid NoC to overcome limitations of BFT and Hoplite. The Hybrid NoC architecture combines BFT and Hoplite NoCs to address layout shortcomings of the BFT NoC, and the multi-flit limitation of the Hoplite NoC. In Hybrid NoC, we use the rings from the Hoplite NoC to provide inter-SLR communication, and we use BFTs to provide intra-SLR communication. Our objective is to interface 32 PEs with the 32 HBM ports. To that end, we use 32 vertical rings from the Hoplite NoC that span the entire chip, which interfaced with 32 PEs are grouped into four rows of BFTs, each with eight PEs. A row of BFT will have eight ports available at its topmost level for interfacing with the 32 rings. So, we use a fan-out and fan-in module to connect one top-level port to four rings. A switch at the topmost level of the BFT can ingress and egress packets into any connected four rings. Similar to Hoplite, the choice of the ring is dictated by **port_id** field of the header flit. For memory packets, the fan-out module chooses the channel determined by the equation (**port_id** $\bmod 4$), whereas for other types of packets, the PEs can choose the ring by varying the **port_id** field of the header.

Each vertical ring has four endpoints for the four rows of BFT, and one more endpoint is added for interfacing with HBM ports through the HBM-NoC interface module. So, the first four endpoints connect to the rows of BFT, and the fifth endpoint connects to the HBM-NoC interface module. However, using all 32 rings will require 16704 SLLs, which is more than available SLLs after accounting for Vitis. Again, we use 24 vertical links with 24 PEs to fit the Hybrid NoC in the chip.

1) Operation: All the switches, fan-in, and fan-out modules in the NoC use the lock system described supporting multi-flit routing. Once injected into the network, the memory packet climbs the BFT, routed towards the ring that connects to the desired HBM port. At the topmost level, the fan-out module injects the packets into the ring determined by the equation (**port_id** $\bmod 4$). Then, the memory packet travels down the ring and reaches the desired HBM AXI port defined by the memory address. There are a few advantages to this topology. By substituting the horizontal links of the Hoplite NoC with mini-BFTs, we ensure deadlock-free operation in the horizontal axis for multi-flit packets without the use of virtual channels. Furthermore, this topology retains the floorplan flexibility of the Hoplite NoC that was absent in BFTs. To break the read request and read response circular dependencies in the vertical rings, we introduce virtual channel support, in only the vertical rings [25].

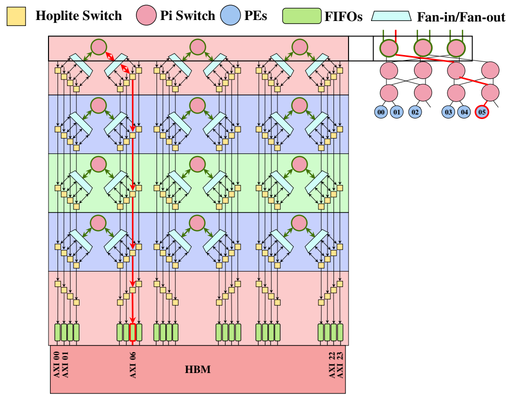


Fig. 4. PEs interfaced to HBM using Hybrid NoC. Modules in the RED region are assigned to SLR0, and it also includes HBM ports. Modules in the BLUE region are assigned to SLR1, and modules in the GREEN region are assigned to SLR2. The path highlighted in red is taken by memory packets injected by PE06 with the memory address set to $60000000h$. For readability, only one row of BFT is shown in the figure.

2) Floorplan: Similar to BFT and Hoplite NoC, we floorplan the Hybrid NoC to span the chip. In Figure 4 we show a top-level view of the floorplan used for Hybrid NoC. The figure shows that the ring NoC reduces the path delay on the loop around the link without spending any additional resources in pipelining. Ring and BFT switches and PEs from the last two rows are assigned to SLR0, whereas two rows of BFT and the corresponding Ring NoC switches are assigned to SLR1, and SLR2 gets one row of BFT. Therefore, 12528 SLLs are required at each SLR crossing, well within the limits.

V. Benchmarking

We evaluate our design using two types of benchmarks:- Synthetic and Application-based. The synthetic benchmarks help establish broad trends in throughput variations by stress-testing the internal crossbar with different traffic patterns. The application-based benchmarks, Blocked Dense Matrix-Matrix Multiplier (DMM) and Sparse Matrix-Vector Multiplier (SpMV), show the effects on throughput for a more realistic traffic pattern.

A. Synthetic Benchmarks

We develop a synthesizable synthetic benchmarking PE that generates different memory access patterns controllable at the runtime, as listed in Table I. The synthesizable PE saturates the HBM by issuing memory transactions every cycle. First, it writes to the HBM every cycle, writing 256MBs of data to the HBM memory. Then, the PE reads the same memory locations it had previously accessed to write, and this is used to verify if the writes have taken place as intended. Finally, after receiving all the read responses from HBM, the PE writes to HBM the total cycle count it took to complete the workload, along with an error bit to indicate a lapse in correctness.

1) Benchmark Design: To generate various synthetic benchmarks, The PE varies the **port_id** based on an addressing policy, base number (N) and the radius (R). The addressing policy determines the relationship between the base number N and the **port_id** that the PE uses for its memory transactions.

TABLE I
Addressing Policies used for Synthetic Benchmarks

Addressing Policy	**Target port_id (P)**
Peer-to-Peer (P2P)	$P = getPortId(N, R)\%24$
Cross-Bank (CB)	$P = getPortId(N+4, R)\%24$
Cross-Stack (CS)	$P = getPortId(N+16, R)\%24$

Whereas the radius defines the range for **port_id** that the PE is allowed to use. Equation 1 is used by the PE to get the **port_id** for a base number N with radius R. It generates R possible values for **port_id** around N.

$$getPortId(N, R) = N + (rand() \bmod R) - (R/2)) \quad (1)$$

For every memory transaction, the PE, depending on the addressing policy, could vary the target **port_id**. With radius 1, the PEs access only one pseudo channel. Whereas, as the radius increases, the number of possible values for **port_id** increases, and the PEs randomly choose a new one for every memory transaction.

Table I shows the formulas used by the PEs to generate the target **port_id** for its memory transactions for various addressing policies. For Cross-Bank addressing policy, each PE will access pseudo channels in the next bank. When the radius is 1, only one pseudo channel is accessed by the PE. With the radius set to 4, PE will access all pseudo channels of the next bank, taking care to avoid Cross-Stack access for underflow and overflow conditions. Similarly, for the Cross-Stack addressing policy, each PE will access pseudo channels in the next stack. When the radius is 1, only one pseudo channel is accessed by the PE. With the radius set to 16, PE will access all pseudo channels of the next stack.

B. Application Benchmark

1) Blocked Matrix Multiplication: We multiply two dense, square matrices, A and B, to compute matrix D in a blocked manner. The output matrix D is broken down into multiple smaller matrices called blocks, and they are computed in parallel by different PEs. By assigning PEs to compute different output blocks in parallel, we exploit the fine-grain parallelism of Matrix-Matrix multiplication. The processing element is described in HLS, and it consists of two sub-modules, a scheduler and compute module. The PE is interfaced to the NoC through two AXIS ports. One AXIS port acts as the master, and it is used to initiate transactions that write to the network, such as write and read request transactions. The other AXIS port acts as the slave, and it is used to read from the network. The scheduler module orchestrates the movement of data to and from the HBM. The compute module is comprised of DSP blocks, on-chip URAM memory, and a register file, which is used to perform the multiply and accumulate operation.

Based on the HLS reported numbers, for a matrix size of 1024 and a block size of 8, a PE takes 601092 cycles to complete its execution. Where 524800 (87%) cycles are compute cycles. The post Place and Route resource utilization for a standalone PE are tabulated in Table II.

TABLE II
POST PLACE AND ROUTE RESOURCE UTILISATION OF A APPLICATION-BASED PEs.

	LUTs	FFs	DSPs	URAMs	BRAMs	Frequency
DMM	5729	4677	192	20	-	320 MHz
SPMV	5608	3651	9	8	32	390 MHz
Synthetic	778	1055	-	-	-	394 MHz

2) Sparse Matrix Vector Multiplication: We also designed a Sparse Matrix-Vector Multiplication, where we multiply a sparse matrix **A** of size $M \times M$ and with a vector **x** of size M resulting in an output vector **b** of size M. In our implementation, the output vector **b** is reused as the input vector **x** for every new iteration of the multiplication. Similar to the DMM, the workload is evenly divided across several PEs, where each PE is assigned to compute some of the output vector elements. Like DMM PE, SpMV PE also consists of two sub-modules, a scheduler and compute module adopted for sparse operations, and it is interfaced to the NoC through two AXIS ports.

The PE operates in four phases:

1) The PE reads and stores the required rows of the sparse input matrix **A** in CSR format and the input vector **x**.
2) PE multiples the stored rows of matrix **A** and the input vector **x**.
3) Computed elements of the output vector are brodcasted to other PEs. In parallel, it receives output vector elements from other PEs. This step is several times based on the user input provided at runtime.
4) The output is written back to HBM.

With an input matrix of dimension 1024×1024, with 65536 non-zero values, and the iteration count set to 16, our optimized PE takes 46289 cycles to complete, wherein 34935 (75%) cycles are compute cycles. The post Place and Route resource utilization for a PE are tabulated in Table II.

C. Hardware Setup

We use the Vitis 2020.2 toolchain to compile the RTL implementation of our designs for the Xilinx Alveo U280 datacentre FPGA. We package our design as a kernel with top-level AXI ports and use Vitis to map the kernel ports to HBM AXI3 ports. Vitis also instantiates and configures PCIe interfaces for communication between the host and the target platform. We use an OpenCL-based host code to load the HBM with required values, execute the kernel, retrieve the result from HBM, and verify the correctness.

VI. RESULTS

A. Experiment methodology

To establish a baseline to compare the effect of the NoCs on throughput, we first interface the benchmarking PEs directly to the HBM ports including the interfacing FIFOs; we call this the noNoC design. Then we repeat the same sweep of experiments with PEs interfaced to HBM using our NoCs.

B. Synthetic Benchmarking

a) What is the effect of Global accesses on throughput?

In Figure 5, we plot the throughput per port against P2P addressing policy with different radii and different burst lengths. The operating frequency determines the achievable throughput for radius 1. Since the synthetic benchmarks without an NoC operate at a higher frequency, they achieve higher throughput than PEs with an NoC. However, the achievable throughput without NoCs quickly drops off as the radii increase, despite the higher operating frequency.

With the single-flit PEs (Figure 5a), both the NoC design improve the attained throughput. Hoplite NoC, despite its limitations, beats the noNoC design. For full global accesses (Radius 24), Hoplite improves the throughput by 5×. However, the BFT offers the highest throughput for all radii larger than 1. Notably, with the radius set to 24, BFT improves the throughput by a factor of 8.6 compared to noNoC.

Similarly, for burst lengths 2, 4, and 8 (Figures 5b to 5d), the attained throughput per port is higher with our NoCs. Hybrid offers the highest throughput per port for all data points barring P2P-8 with burst lengths 4 and 8, where BFT beats the Hybrid NoC. However, the results are mixed for burst length 16. The BFT loses to noNoC design for all radii, barring 24. The Hybrid NoC offers higher throughput than noNoC design only for radii 2 and 24, with a speedup of 1.26 for radius 24.

TABLE III
RESULTS FROM SPMV BENCHMARKING EXPERIMENT

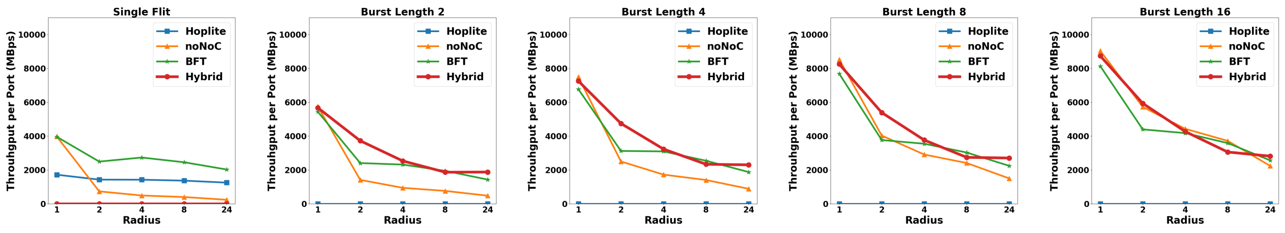
Benchmark	No. NZV	NoC Type	Freq (MHz)	Tput/port (MBps)	Tput Util.(%)	Runtime (ms)
dw8192	**41746**	**noNoC**	174	2624.23	49	68802
		Hybrid	236	6403.15	88	51019
		BFT	203	5677.1	91	49030
epb	**95053**	**noNoC**	174	2512.03	47	117189
		Hybrid	236	6615.37	91	87412
		BFT	203	5831.25	94	85877
t2d_q9	**87025**	**noNoC**	174	2647.07	47	96241
		Hybrid	236	6494.53	91	67029
		BFT	203	5722.12	94	68309

b) What is the effect of Cross-bank and Cross-stack access on throughput?

In Figure 6a we plot the throughput per port against the radius for single-flit designs with cross-bank addressing policy, and we plot the throughput versus burst length for multi-flit design with cross-bank addressing policy in Figure 7. From Figure 6a, it can be observed that across all data points, the BFT-SF NoC outperforms the noNoC-SF design. The throughput attained by noNoC-SF design drops by 6× when the radius is increased to 4, and BFT-SF improves this throughput by 5.6×. However, Hoplite-SF performs poorer

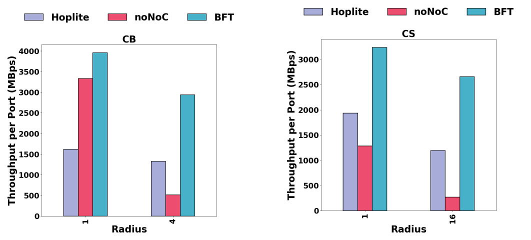
TABLE IV
RESULTS FROM DMM BENCHMARKING EXPERIMENT

Matrix Size	NoC Type	Frequency (MHz)	Tput/port (MBps)	Tput Utili. (%)	Runtime (ms)
17920	**noNoC**	254	4217	54.42	44462.4
	Hybrid	256	5393.29	69.03	84055
	BFT	251	5376.3	70.18	70431.9
8064	**noNoC**	254	4220	54.44	4043.38
	Hybrid	256	5417.47	69.34	7631.69
	BFT	251	5310.94	69.33	6372.62
2048	**noNoC**	254	4301.81	48.87	0.75
	Hybrid	256	5322	66.5	0.71
	BFT	251	5115.55	70	0.68



(a) Single Flit (b) Burst Length 2 (c) Burst Length 4 (d) Burst Length 8 (e) Burst Length 16

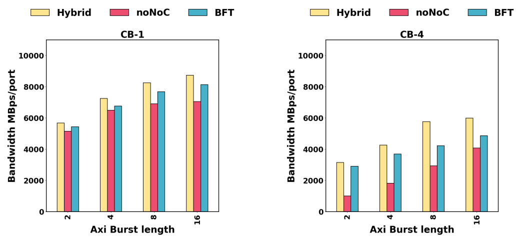
Fig. 5. Throughput per port versus P2P addressing policy for different burst lengths



(a) Throughput vs Radius for single-flit designs with Cross Bank addressing policy

(b) Throughput vs Radius for single-flit designs with Cross stack addressing policy

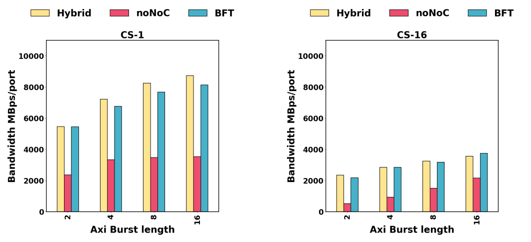
Fig. 6. Throughput results for single-flit networks with Cross bank and Cross stack addressing policy



(a) Throughput vs Burst Length with CB addressing policy with Radius 1

(b) Throughput vs Burst Length with CB addressing policy with Radius 4

Fig. 7. Throughput results with Cross bank addressing policy



(a) Throughput vs Burst Length for with CS addressing policy with Radius 1

(b) Throughput vs Burst Length for with CS addressing policy with Radius 16

Fig. 8. Throughput results with Cross Stack addressing policy

than the noNoC-SF. Despite the limit on pending read requests, Hoplite-SF improves the throughput by 2.5$\times$ compared to noNoC-SF for radius 4.

Both the BFT-MF and Hybrid-MF NoCs surpass the throughput attained by the noNoC-MF design, where the Hybrid-MF outperforms both the BFT-MF and noNoC-MF design, as seen in Figure 7. For Cross-Bank addressing policy with radius 4, Hybrid NoC improves the throughput by a factor of 3.14 $\times$ (burst length 2). However, the gap between noNoC-MF and the NoCs narrows with increased burst length. As a result, the throughput speedup delivered by the Hybrid NoC drops from 3.14$\times$ for burst length 2$\times$ to 1.46$\times$ for burst length 16.

We observe a similar trend for Cross-Stack addressing policy as well, as seen in Figures 6b, and 8. In both the single-flit and multi-flit design, the throughput attained by noNoC-SF drops by a factor of 0.3$\times$ for Cross-Stack accesses compared to P2P-1. For single-flit designs (Fig. 6b), both the Hoplite-SF and BFT-SF deliver higher throughput than noNoC-SF, with BFT-SF emerging as a clear winner. Unlike the Cross-Bank addressing policy, Hoplite-SF outperforms the noNoC-SF, despite the limit on pending read requests. With radius 1, Hoplite-SF improves the throughput by 1.5$\times$, and the BFT-SF improves it by a factor of 2.5$\times$. With radius 16, the throughput gains are higher. Hoplite-SF improves the throughput by 4.4$\times$, and the BFT-SF improves it by a factor of 9.8$\times$.

Likewise, for multi-flit designs (Fig. 8) use of NoCs improve the attained throughput. Unlike Cross-Bank policy, there is no clear winner between the NoCs. Hybrid-MF NoC marginally outperforms BFT for burst lengths up to 8, whereas BFT-MF outperforms for burst length 16. With burst length 2, Hybrid-MF improves the throughput by 4$\times$, and when the burst length is increased to 16, the throughput speedup drops to 1.7$\times$.

C. *Sparse Matrix Vector Multiplication*

We used 24 PEs to compute SpMV for *dw8192*, *epb*, and *t2d_q9* benchmark matrices for this experiment, tabulating the results in Table III. We observed that both the BFT and Hybrid NoC offer up to 1.40$\times$ and 1.43$\times$ speedup in runtime, respectively, when compared to the noNoC design. Both the NoC perform better as the sparsity increases. However, the BFT outperforms the Hybrid for all benchmarks barring the t2d_q9 benchmark. The Hybrid NoC's runtime is marginally lower for the t2d_q9 benchmark. BFT offers the highest bi-section bandwidth of $O(N)$ for PE to PE communication. In the Hybrid NoC, we replace the top two levels of the Pi switches with several ring NoC, lowering the bi-section bandwidth for those two levels. Similarly, Hybrid NoC offers the highest throughput per port, as reported by Vitis. But, if we consider the throughput utilization per port corresponding to the operating frequency, BFT NoC beats the Hybrid NoC,

TABLE V
MAXIMUM OPERATING FREQUENCIES (MHZ) FOR DESIGNS USED IN OUR EXPERIMENT

NoC Type	No PE	Synthetic	DMM	SpMV
BFT-SF	327	300	-	-
Hoplite-SF	340	306	-	-
noNoC-SF	-	340	-	-
BFT-MF	301	272	251	203
Hybrid-MF	297	290	256	236
noNoC-MF	-	300	254	174

and both the NoCs roughly improve the throughput utilization per port by at least 1.92×.

D. Dense Matrix Matrix Multiplication

For DMM benchmarking experiment, we used 16 PEs with different input matrix sizes of dimensions 2048×2048, 8064×8064 and 17920×17920, and the results are tabulated in Table IV. We observed that both the BFT and Hybrid NoC improve the throughput utilization per port by at most 1.29× vs the NoCs. However, the latency of the multiplication is significantly higher than the noNoC design. In the worst case, PEs with Hybrid NoC takes 1.9× the time taken to complete a workload of matrix size 17920 than without any NoC. We can infer that with DMM PEs, the throughput increase obtained by using NoCs is offset by the additional cycles taken for the flits to travel through the NoC. Furthermore, the table shows that the runtime performance with the NoCs gets poorer with the increase in the matrix size. As the matrix size increases, the NoC gets congested with packets from all the PE, and as a result, the packets spend more time in the network, further worsening the latency.

E. Design Operating Frequencies

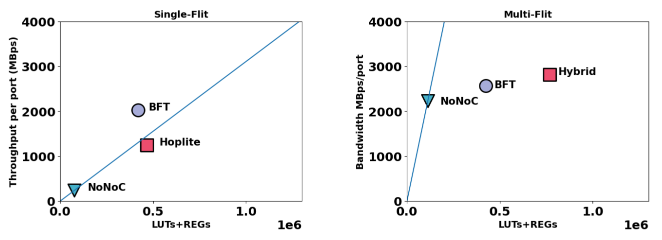
In Table V, we tabulate the maximum operating frequencies of our single-flit (SF) and multi-flit (MF) NoC designs.

Without any PE, the Hoplite-SF has the highest operating frequency of 340 MHz. This can be attributed to the floorplan-friendly torus topology of the Hoplite. On the other hand, BFT and Hybrid suffer from the complexities of floorplaning a hierarchical topology. This leads to higher congestion and lower frequency.

The frequency of BFT-MF without PEs is marginally higher than the Hybrid-MF. However, for all PEs the frequency of BFT-MF drops below the Hybrid due to the complexities of its floorplanning . While the Hybrid-MF also needs nested floorplanning for its BFT-MF rows, it is only three levels of nesting and contained in single SLR. With BFT-MF, we need five levels of nesting spread across 3 SLRs, worsening the congestion when PEs with high resource utilization such as SpMV is interfaced with the network.

Synthetic benchmarking PEs consume fewer resources than SpMV or DMM PE. Therefore, they have a higher operating frequency than their counterparts with an NoC. However, for SpMV PE without any network, the Vivado struggles to place and route the PEs in SLRs other than SLR0 resulting in lower frequency.

F. Resource Utilization



(a) Throughput vs LUTs+REGs for P2P-24, for single-flit network

(b) Throughput vs LUTs+REGs for P2P-24, for multi-flit network, burst length 16

Fig. 9. Throughput vs LUTs+REGs

In Figure 9, we plot the attained Throughput per port for P2P-24 addressing policy vs resource utilization for our designs. From Figure 9a, it can be observed that the BFT-SF offers better Throughput per LUT+REG spent than noNoC-SF and Hoplite-SF. However, the observed advantage is diminished for the multi-flit design, as shown in Figure 9b. While the multi-flit network offers higher throughput per port, the effectiveness of our networks drops with the increase in burst length. Because, with the increase in burst length, any throughput gains offered by the network is offset by the additional clock cycles consumed by it due to increased contention. Furthermore, the presence of Virtual channels in Hybrid NoC requires buffers with pipelined inputs are in every ring switch, resulting in higher resource utilization. It consumes 1.8× the LUTs+REGs consumed by BFT-MF NoC without a commensurate increase in throughput. Consequently, both the BFT-MF and Hybrid-MF offer less Throughput per LUT+REG than noNoC-MF design for P2P-24 with burst length 16.

G. Key Takeaways

To summarise, when the memory accesses made by the PE do not cross the bank boundary, and if they can be fit into SLR0, then no NoCs are required. However, for a chip-spanning design with PE accesses to all pseudo channels, NoCs improve the throughput. Hybrid-MF offers the best raw throughput gains for multi-flit designs. However, BFT-MF offers a better throughput resource trade-off. For single flit designs, BFT is the preferred topology for performance.

VII. CONCLUSIONS

This paper demonstrates the use of FPGA overlays NoCs to improve HBM bandwidth distribution and utilization in a multi-die FPGA like Xilinx Alveo U280. We benchmark the HBM using synthetic and real workloads and show that due to HBM's internal crossbar, the bandwidth drops by 3-10× with global access. Furthermore, our experiments show that using Overlay NoCs improves the HBM throughput per port. Our experiments show that Overlay NoCs can improve the throughput by 1.26× for synthetic benchmarks and up to 1.4× for SpMV workloads. For DMM, the increased in contention in the NoCs, offset the throughput gain offered by the NoC, degrading overall throughput performance..

References

[1] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[2] M. Besta, D. Stanojevic, J. de Fine Licht, T. Ben-Nun, and T. Hoefler, "Graph processing on fpgas: Taxonomy, survey, challenges," *CoRR*, vol. abs/1903.06697, 2019.

[3] A. Becher, D. Ziener, K. Meyer-Wegener, and J. Teich, "A co-design approach for accelerated sql query processing via fpga-based data filtering," in *2015 International Conference on Field Programmable Technology (FPT)*, pp. 192–195, 2015.

[4] P. Papaphilippou, J. Meng, and W. Luk, "High-performance fpga network switch architecture," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, (New York, NY, USA), p. 76–85, Association for Computing Machinery, 2020.

[5] K. Tran, P. Silvestri, B. Isaacson, B. Daellenbach, and C. Browy, *Start Your HBM/2.5D Design Today*, 2016.

[6] Xilinx, *Alveo U280 Data Center Accelerator Card Data Sheet*, 2020.

[7] Xilinx, *AXI High Bandwidth Memory Controller v1.0 LogiCORE IP Product Guide*, 2020.

[8] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Benchmarking high bandwidth memory on fpgas," 2020.

[9] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "Hbm connect: High-performance hls interconnect for fpga hbm," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, (New York, NY, USA), p. 116–126, Association for Computing Machinery, 2021.

[10] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on fpgas," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 111–119, 2020.

[11] Xilinx, *Getting Started with Alveo Data Center Accelerator Cards*, 2020.

[12] Xilinx, *UG574 UltraScale Architecture Configurable Logic Block*, 2021.

[13] Xilinx, *Alveo U280 Data Center Accelerator Card User Guide*, 2020.

[14] N. Kapre, "Deflection-routed butterfly fat trees on fpgas," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, 2017.

[15] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, 1985.

[16] G. S. Malik and N. Kapre, "Enhancing butterfly fat tree nocs for fpgas with lightweight flow control," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 154–162, 2019.

[17] N. Kapre and J. Gray, "Hoplite: Building austere overlay nocs for fpgas," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, 2015.

[18] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Computing*, vol. 1, no. 4, p. 187–196, 1986.

[19] N. Kapre and J. Gray, "Hoplite: A deflection-routed directional torus noc for fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, Mar. 2017.

[20] J. Gray, *2GRVI Phalanx: A 1332-core RISC-V RV64I Processor Cluster Array with an HBM2 High Bandwidth Memory System, and an OpenCL-like Programming Model, In a Xilinx VU37P FPGA [WIP Report]*, 2018.

[21] E. Fragkakis, "Deadlock avoidance with virtual channels," Master's thesis, 2009.

[22] M. N. Emas, A. Baylis, and G. Stitt, "High-frequency absorption-fifo pipelining for stratix 10 hyperflex," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 97–100, 2019.

[23] Xilinx, *High-Performance, Lower-Power Memory Interfaces with the UltraScale Architecture WP454*, 2015.

[24] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High bandwidth memory on fpgas: A data analytics perspective," 2020.

[25] J. W. Dally and B. Towles, *Principles and practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2003.