2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) | 979-8-3315-0281-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/FCCM62733.2025.00062

NoH: NoC Compilation in High-Level Synthesis

Huifeng Ke*, Sihao Liu*, Licheng Guo†, Zifan He*,
Linghao Song*, Suhail Basalama*, Yuze Chi†, Tony Nowatzki*, Jason Cong*,
*University of California, Los Angeles, †RapidStream Design Automation, Inc.,
jakeke@cs.ucla.edu,

***Abstract*—In FPGAs, high communication latency in multi-die chips has driven the integration of hardened networks-on-chip (NoCs) in commercial devices. However, for programming FPGAs with high-level synthesis (HLS), existing tools only provide low-level cumbersome abstractions, and only work for offloading memory accesses. Furthermore, these abstractions remain inaccessible to programmers due to their reliance on placement knowledge. While automatically leveraging the NoC without manual intervention is ideal, it poses several challenges: 1. Managing the trade-off in resource utilization between the hard NoC and the Programmable Logic (PL). 2. Allocating limited hard NoC resources between different communication in the designs. 3. Aligning hard NoC and PL placement even though the actual PL placement cannot be determined beforehand.**

We address these challenges by developing NoH, the first HLS flow that automates hard NoC offloading. First, we develop a formal NoC-aware placement algorithm that leverages integer linear programming (ILP) and considers the first two challenges for offloading external memory accesses and latency-insensitive communication between modules. Then, we arrange the ports synergistically with PL modules via a port-affinity model that approximates the PL placement. Finally, NoH is integrated into an end-to-end HLS flow and evaluated on 4 workloads with diverse communication patterns. NoH gains 20% FPGA frequency over AMD tools by leveraging the hard NoC. Compared to AutoBridge [1], a recent high-level physical synthesis technique that optimizes frequency but does not consider the hard NoC, NoH never fails place-and-route by offloading inter-die crossings (AutoBridge fails in 31% of workload configurations tested) and is faster (6%) for the rest.

I. Introduction

HLS is gaining prominence for enabling productive programmability for FPGA systems [2]. It has shown success in many application domains, including deep learning [3]–[9], video transcoding [10], graph processing [11]–[22], and genome sequencing [23]–[29].

Despite these successes, communication on FPGAs is increasingly burdensome. To help address the communication bottleneck, major FPGA vendors such as AMD [30] (Figure 1), Intel [31], and Achronix [32], have introduced a system-level hard network-on-chip (NoC). Hard NoCs increase the bandwidth per wire due to higher frequency, facilitate efficient time-multiplexing, and guarantee interconnect timing closure.

However, it can be difficult to exploit the benefits of hard NoC, especially for HLS users. Users would need knowledge of coarse-grained placement at design time to exploit hard NoC bandwidth and achieve high frequency at the same time. If NoC port placement is not provided, the place and route (P&R) tool may not be able to meet the NoC bandwidth

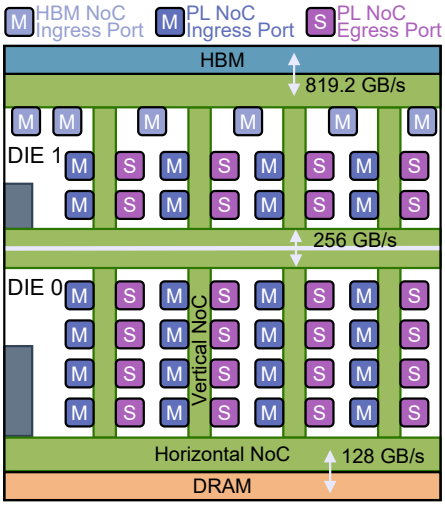


Fig. 1: A sketch of the hard NoC in the AMD Versal VHK158 FPGA board. A subset of NoC ports are shown.

constraints due to competing PL placement. That limitation undermines the hardware abstraction HLS brings. In addition, it is not trivial to decide what to offload on to the hard NoC. In fact, commercial FPGA HLS compilers do not yet support instantiating NoC within HLS designs, except for external memory. Even then, the current P&R tool cannot optimize the placement of programmable fabric cooperatively, encumbering the potential resource savings and frequency gains of utilizing the hard NoC.

An ideal approach is to automatically infer hard NoC usage, based on communication patterns in the HLS program without any programmer's help. However, there are several challenges that must be addressed. First, we must balance the use of hard NoC and PL resources to achieve optimal performance. Overusing the hard NoC can impact latency and throughput, while relying too much on the PL resources left the hard NoC wasted and could make P&R difficult. Second, we need to trade-off using the limited hard NoC resources for different communication in the design (i.e. for memory accesses versus for communication between modules[1]. Third, the arrangement of NoC ports needs to be synergistic with the PL placement, otherwise congestion can occur near the hard NoC ports. The key problem is that the actual PL placement cannot be

[1]In this paper, "**module**" refers exclusively to RTL modules.

2576-2621/25/$31.00 ©2025 IEEE
DOI 10.1109/FCCM62733.2025.00062

determined before selecting which communication is offloaded to the NoC, creating a cyclic dependence.

We address these challenges in our technique: NoH, the first automated hard NoC compilation framework for FPGA HLS. Furthermore, to the best of our knowledge, no prior HLS or RTL technique can decide which communication channels to offload to soft logic vs hard NoC and how. NoH can be integrated into commercial HLS tools and does not need modification to the input C++ HLS program, thus improving frequency while preserving the high-level programming. NoH consists of two main algorithms, both formulated in integer linear programming (ILP): The first is a **NoC-aware Coarse-grained Placement** algorithm, specifically prioritizing the memory IOs for the hard NoC while balancing the PL inter-die wire utilization. The second is a **Hard-NoC Offloading** algorithm that determines both which latency-insensitive communication to offload and how to arrange the NoC ports synergistically with the PL placement. We resolve the cyclic dependence by employing a port-affinity model that is a proxy for how PL modules will be placed. These are integrated as a multistage optimization algorithm to permit tractable compilation.

We compare NoH against two strong baselines which do not use the hard NoC: the AMD Versal tools, as well as AutoBridge [1], the previous state-of-the-art high-level physical synthesis (HLPS) tool that optimizes design frequency. It couples physical design with HLS scheduling and automatically pipelines inter-die paths. Using 16 configurations of four workloads that represent unique communication scenarios, our evaluation demonstrates a 20% frequency improvement over Versal Tools by eliminating problematic congestion and high-delay inter-die routes. While AutoBridge can sometimes reach similarly high frequencies, it often fails due to congestion on die boundaries (31% of configurations). NoH significantly reduces inter-die crossings, which enables consistent compilation without failure and achieves >226 MHz on average.

Contributions:

1) A technique, NoH, for automatically utilizing hard NoC in HLS designs to minimize inter-die crossings and improve frequency.
2) An algorithm for cooperative P&R for the PL and hard NoC leveraging integer linear programming (ILP).
3) Evaluation using commercial tools of four real-world workloads with sixteen configurations to show the frequency increase from employing the hard NoC.

II. Background

Versal NoC: The Versal NoC [30] is a packetized network with 128 bit width running at 1 GHz. Figure 1 depicts a sketch of the hard NoC in VHK158. The raw throughput of each unidirectional link is 16 GB/s. The Versal NoC adopts a mesh-like topology, but with horizontal links positioned at the top and bottom of each die. The Versal NoC provides unified access to all hard and soft components on the device using NoC ingress ports and egress ports. The NoC ingress ports issue requests to the network from the PL, whereas the NoC egress ports service requests from the network to the PL. The PL can access the NoC ports with decoupled clock domains. Versal NoC implements virtual channels, up to 8 per link. NoH uses virtual channels to prevent deadlocks.

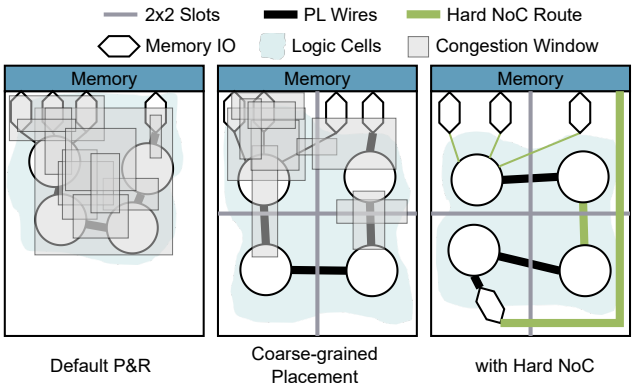


Fig. 2: A toy example illustrating the improvement of using hard NoC. It facilitates a better placement with half of the slot-crossings. Hard NoC further offloads scarce inter-die wires, balances utilization, and reduces congestion.

Hard NoC Integration: Using the hard NoC requires significant programmer effort. To explain, AMD Vivado software provides two IPs to use the hard NoC, the AXI NOC IP for the AXI memory-mapped protocol, and the AXIS NOC IP for the AXI4-Stream protocol. Configuring the two IPs requires the low-level Vivado IP Integrator (IPI) flow, which is not the conventional tool for HLS users. Users need to specify the NoC connectivity and bandwidth requirement. The Vivado's NoC compiler statically determines NoC routes. However, users should provide the placement of NoC ports. Otherwise, the NoC compiler could arbitrarily place the ports, but does not guarantee meeting both bandwidth requirements and high frequency. Our work, NoH, fills the gap by automatically determining the communication to offload to the hard NoC and optimizing the placement of NoC ports.

III. Motivation & Overview

A. Motivational Example

Figure 2 uses a toy example to illustrate the benefits of the hard NoC. The default P&R tool will place memory IOs and modules near memory, resulting in high congestion and underutilized resources. With the coarse-grained placement technique from HLPS tools like AutoBridge, the minimum resource utilization constraint enforces a more balanced placement. The device is divided into a grid of conceptual slots. Tasks are placed in slots while minimizing crossings between slots. However, for large designs, there can be many slot crossings between dice, which often become critical timing paths. Moreover, when the design uses a lot of memory IOs, HLPS tools can still cause high congestion near the memory.

With the hard NoC, we can find a better placement with fewer crossings by placing a memory IO in a distant slot. In

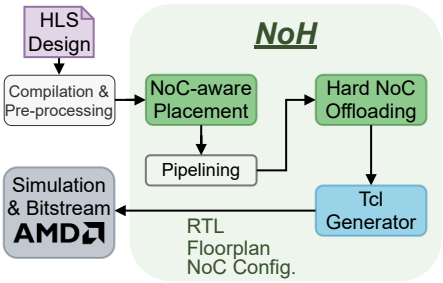


Fig. 3: Our approach to automatically utilize hard NoC in HLS designs.

addition, we can use the remaining bandwidth to offload the rest of the inter-die crossings.

B. Overview of NoH Framework

Shown in Figure 3, NoH has been integrated into a commercial HLS flow. The input of NoH is task-parallel dataflow programs described in TAPA [33], which are compiled to RTL and preprocessed to obtain dataflow graphs. An HLS-level **task** (function) is a user-defined grouping of operations (RTL **modules**). We first run the NoC-aware coarse-placement ILP. It places memory IOs in the hard NoC and tasks in user-defined slot granularity, where **inter-slot** crossings are minimized and then pipelined. We use the remaining hard NoC bandwidth to selectively replace **inter-die** pipelines, while considering how to place NoC ports synergistically with the likely placement of PL modules. We prioritize memory IOs because they could have a larger impact on the global inter-die crossings as illustrated in Section III-A. Alternatively, users have the flexibility to allocate all hard NoC bandwidth for inter-die crossings. The final output is a NoC-enhanced RTL, a coarse-grained placement, and the hard NoC connectivity and QoS configurations. Through NoH, all inter-die crossings are either pipelined or offloaded to the hard NoC.

IV. NoH ILP Formulation

Representation: Both HLS programs and the hard NoC are described in graphs. The dataflow graph (DFG) is a directed graph, where nodes are tasks and edges inter-task communication including memory IOs. The hard NoC graph (HNG) describes the NoC topology, where edges are the NoC links, and nodes include NoC ingress ports, egress ports, and switches.

Algorithm Flow: Coarse-grained placement of DFG involving HNG is a cooperative P&R process. We minimize crossings between slots while ensuring the offloaded memory IOs and the latency-insensitive communication fit within the hard NoC's bandwidth. It is accomplished in two stages, NoC-aware placement and Hard NoC Offloading.

- **Stage 1. NoC-Aware Placement** Here we assign DFG nodes to slots, while memory IOs are mapped to the HNG (assigned to NoC ingress ports). We overwrite the default placement of memory IOs so that they can be distributed across the device through the hard NoC, if that leads to a significant reduction in crossings.
- **Stage 2. Hard NoC Offloading** Here we select some of the latency-insensitive slot crossings that are inter-die to be further offloaded to the HNG given enough NoC bandwidth. We optimize their NoC ingress and egress ports placement to be synergistic to PL placement.

In the following subsections, we first introduce the definitions used by ILP equations. Next, we explain the hard NoC P&R constraints shared by the two ILP stages, followed by the details of each stage. Section IV-A and all equations can be skipped unless readers want to understand the precise ILP formulation.

A. DFG Definitions for ILP equations

- $DFGedges$: edges in DFG to be placed. Each edge, e, has a source slot and a sink slot.
- $NoCingress$: all NoC ingress ports. $NoCingress_e$ are the ingress ports in the source slot of e.
- $NoCegress$: all NoC egress ports. $NoCegress_e$ are the egress ports in the sink slot of e.
- $NoCswitches$: all NoC switches.
- $NoClinks$: all NoC links.

B. Hard NoC P&R Constraints

Given the bandwidth, the source slot and the sink slot of each DFG edge, we introduce the hard NoC P&R constraints to ensure legal offloading by enforcing that each NoC ingress or egress port handles at most one edge (as the Versal NoC does not time-multiplex the NoC ports), and no link exceeds its bandwidth capacity. We first define the following ILP variables **for each** $e \in DFGedges$:

- $a_{e,p}$: (binary) e uses NoC *ingress* port, p, as its source.
- $b_{e,p}$: (binary) e uses NoC *egress* port, p, as its sink.
- $f_{e,l}$: (binary) e uses NoC link l for its *forward* traffic
- $r_{e,l}$: (binary) e uses NoC link l for its *return* traffic
- n_e: (binary) e is not mapped to the HNG.

Choosing NoC Ports: This constraint ensures each DFG edge is mapped to exactly one NoC ingress port and one egress port within the same slot, or it is not mapped. The summation of all candidate port variables should equal to 1 if the edge is mapped (meaning n_e is 0).

$$\forall e \in DFGedges: \sum_{p \in NoCingress_e} a_{e,p} + n_e = 1, \quad (1)$$

$$\sum_{p \in NoCegress_e} b_{e,p} + n_e = 1, \quad (2)$$

NoC Ports Non-overlap: This constraint ensures each NoC ingress and egress port supports at most one DFG edge. This should be updated if the hard NoC ports support time-multiplexing.

$$\sum_{e \in DFGedges} a_{e,p} \leq 1, \quad \forall p \in NoCingress \quad (3)$$

$$\sum_{e \in DFGedges} b_{e,p} \leq 1, \quad \forall p \in NoCegress \quad (4)$$

NoC Routing for Forward Traffic: Flow conservation ensures a legal NoC route for each edge's forward traffic from source to sink. Each selected ingress port has exactly one outgoing flow (`.succ()`) and no incoming flow (`.pred()`), and vice versa for egress ports. The intermediary links must have a conserved flow.

$$\forall e \in DFGedges: \tag{5}$$

$$\sum_{l \in p.succ()} f_{e,l} = a_{e,p}, \sum_{l \in p.pred()} f_{e,l} = 0, \forall p \in NoCingress_e, \tag{6}$$

$$\sum_{l \in p.pred()} f_{e,l} = b_{e,p}, \sum_{l \in p.succ()} f_{e,l} = 0, \forall p \in NoCegress_e, \tag{7}$$

$$\sum_{l \in p.pred()} f_{e,l} - \sum_{l \in p.succ()} f_{e,l} = 0, \quad \forall p \in NoCswitches. \tag{8}$$

NoC Routing for Return Traffic: If the DFG edge is a memory IO, it needs a route for the return traffic. We apply the same constraints for forward traffic to $r_{e,l}$ but swapping $NoCingress_e$ and $NoCegress_e$.

Prevent Deadlocks Using Virtual Channels: This constraint ensures that the number of traffic channels on a link does not exceed the number of virtual channels available, VC. We need to assign each edge's traffic to a unique virtual channel to conservatively prevent deadlocks [34] and head-of-line blocking between different traffic channels.

$$\sum_{e \in DFGedges} f_{e,l} + r_{e,l} \leq VC, \quad \forall l \in NoClinks \tag{9}$$

Bandwidth: This constraint guarantees that the aggregated bandwidth mapped to each NoC link does not exceed its capacity. For memory IOs, the forward traffic bandwidth (bw_f) includes write data, write address, and read address, while the return traffic bandwidth (bw_r) includes read data and write responses. For latency-insensitive communication, the address and return bandwidth are zero. The capacity of the NoC links is defined by a constant C. This constraint guarantees that the user-specified bandwidths in the original design are met.

$$\sum_{e \in DFGedges} f_{e,l} \cdot bw_f + r_{e,l} \cdot bw_r \leq C, \forall l \in NoClinks. \tag{10}$$

C. NoC-aware Placement

This stage places DFG nodes in coarse-grained slots and memory IOs in the hard NoC, while minimizing inter-die crossings. Similar to previous HLPS tools [1], we use a multi-stage ILP for placement, where we incrementally make finer-grained decisions until we have reached the user-defined slot granularity. At each step, we logically divide the FPGA device in half, either horizontally or vertically. For example, we split 2 times to get 2x2 slots. We place each $DFGop$ in one of the two new slots. The placement part of ILP is omitted for brevity, and we explain our hard NoC extension.

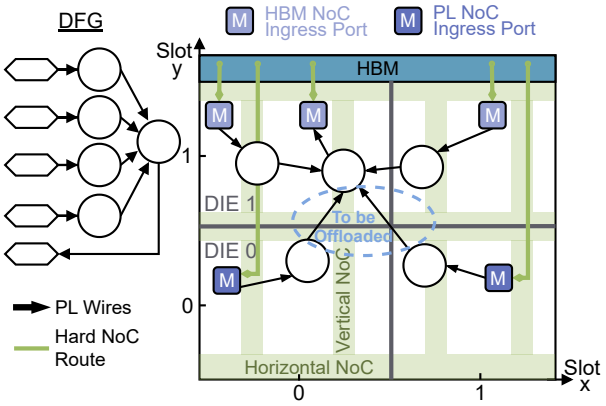


Fig. 4: NoC-Aware Placement ILP. DFG nodes are placed in the 2x2 coarse-grained slots. Memory IOs are offloaded to the hard NoC to reduce slot crossings and balance utilization of both dice.

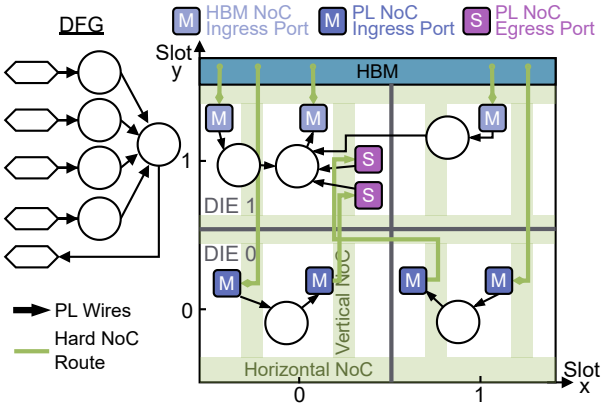


Fig. 5: Hard NoC Offloading ILP. Additional pairs of NoC ports replace the inter-die PL wires. Timing constraints are relaxed by isolating each slot.

Without considering the hard NoC, a trivial heuristic used by prior works is to place IOs near memory. However, the hard NoC allows distribution of memory IOs across the device to reduce inter-die crossings and congestion near memory.

Figure 4 illustrates this intuition using an example DFG. We split twice to place each task in one of the four slots. During each split, a legal hard NoC mapping is determined. Due to the PL resource constraint, two tasks are placed in the lower slots. Thus, their memory inputs are also placed in the lower slots to reduce slot crossings and congestion near HBM.

On-chip Resources Constraint: This constraint ensures the area of all DFG nodes placed in each slot does not exceed the capacity. We consider all types of on-chip resources (except hard NoC)—LUTs, registers, BRAMs, URAMs, and DSPs.

Placing Memory IOs in Hard NoC: We use the constraints in Section IV-B to map only memory IOs to the hard NoC. We must map all memory IOs (i.e., constraining all n_e to

0) because all external memory accesses go through the hard NoC in Versal FPGAs.

Objective Function: Minimizes all crossings between slots. $e.width$ is the bit width of the DFG edge e. $mahattan(e)$ returns the Manhattan distance between the source and sink slots.

$$Minimize \sum_{\forall e \in DFGedges} e.width \cdot manhattan(e) \quad (11)$$

D. Hard NoC Offloading

After the NoC-aware Placement stage, this stage cuts down the latency-insensitive inter-die crossings using the **remaining bandwidth** in the hard NoC. It has three objectives to achieve in the specified order (*the result of the earlier objective is set as a constraint for the next*). The ILP will return the selected inter-die DFG edges to offload to the hard NoC. The third objective is called PL-NoC Cooperative Placement, detailed in Section IV-E. Here, we present the first two objectives.

Figure 5 shows the changes after stage 1, where the two inter-die wires in the lower slots are replaced with two pairs of connected hard NoC ports. Lower utilization of inter-die wires reduces the congestion near die boundaries.

Fixing Memory IOs Placement: We apply the constraints from Section IV-B and fix them to the memory IO placement derived from the previous stage, which means that the user-defined memory bandwidth requirements are preserved during further offloading.

Placing Inter-die Edges: We use the constraints in Section IV-B to map inter-die crossings this time. We create ILP variables only for the inter-die DFG edges. Due to limited NoC bandwidth, only a subset will be selected.

First Objective: The most important objective is to minimize inter-die crossings. $levels(e)$ returns the number of slots from the source to the sink of e.

$$Minimize \sum_{e \in DFGedges} n_e * e.width * levels(e) \quad (12)$$

Second Objective: The second objective is to minimize the number of mapped hard NoC links. The hard NoC's latency increases with the number of NoC links it goes through. We want to keep the latency overhead low.

$$Minimize \sum_{e \in DFGedges} \sum_{l \in NoClinks} f_{e,l} + r_{e,l} \quad (13)$$

E. PL-NoC Cooperative Placement

After achieving the first two objectives, our formulation produces a legal hard NoC P&R solution. However, the NoC ports placement may be suboptimal relative to the PL placement, potentially causing congestion near NoC ports. Unfortunately, since PL placement is unknown at this stage, but NoC ports placement must be fixed before PL placement, it creates a cyclic dependence where direct optimization is not feasible.

Instead, our approach is to use a proxy model based on port affinity, where higher affinity indicates that ports should

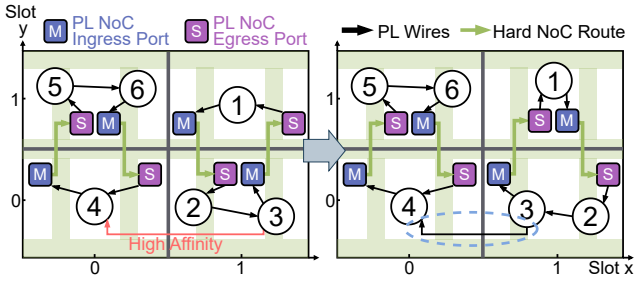


Fig. 6: PL-NoC Cooperative Placement. We use a proxy model based on port affinity to align NoC ports with PL placement. Note the PL wire from node 3 to 4 becomes shorter.

be adjacent because their consumers/producers are closely related. We define the affinity using the minimum number of DFG nodes between two DFG edges. A lower number of nodes means that two edges are more related, and we should place their NoC ports closer. For example, as shown in Figure 6, directly connected nodes 3 and 4 have high affinity, so aligning their NoC ports (node 4's egress port and node 3's ingress port) yields an aligned PL placement and shorter wiring. We define $affinity(e0, e1)$ to return the affinity between two edges. Dividing the distances between NoC ports by their affinity indicates the importance of placing them together.

To calculate the distance, it is essential to use the physical locations of NoC ports, which are arranged in columns distinct from the mesh-like NoC topology. We define two helper functions to calculate the physical distance between NoC ports: $dist(e0, e1)$ returns the Manhattan distance between the NoC ports of two DFG edges (can be limited to pairs within the same slot to keep the ILP tractable), and $dist(e)$ measures the Manhattan distance between an edge's NoC ingress and egress ports for inter-slot alignment. We minimize the total distance with tunable weights, W_1 and W_2.

Third Objective:

$$D_{pairs} = \sum_{(e0,e1) \in \binom{DFGedges}{2}} \frac{dist(e0, e1)}{affinity(e0, e1)} \quad (14)$$

$$D_{src\_sink} = \sum_{e \in DFGedges} dist(e) \quad (15)$$

$$Minimize\ W_1 * D_{pairs} + W_2 * D_{src\_sink} \quad (16)$$

V. Implementation

NoH leverages TAPA [33], an efficient C++ front-end API for compiling HLS designs and extract DFGs. ILP is implemented with the Python PULP package and the Gurobi solver. The total runtime on average is within 30 minutes. Users of NoH must specify slot granularity, HNGs, and their design's memory IO configurations (including the designated external memory bank, read bandwidth, and write bandwidth). We run AutoBridge to pipeline the inter-die crossings. RTL is transformed to expose the mapped inter-die connections as top-level ports so they can be connected to the hard NoC in the Vivado's IPI flow.

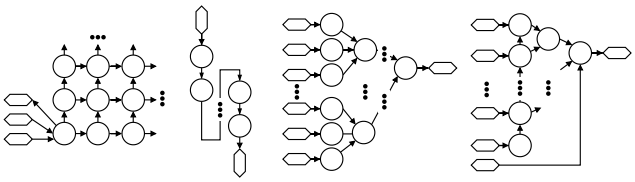


Fig. 7: Simplified workload topology, only capturing memory IOs and computation nodes.

We use Tool Command Language (Tcl) to create a unified block design capable of both simulation and bitstream generation. Our Tcl script instantiates the embedded ARM core acting as the host. The host generates the design's clock and reset, sends control signals to the design, and verifies the results in external memory. For RTL simulation, we use the AMD Vivado Simulator (xsim) and Versal CIPS Verification IP. NoC RTL simulation is within ± 5% of hardware [35].

VI. Methodology

Workload Selection: In order to stress different kinds of communication patterns over the hard NoC, we choose four workloads with unique topologies, as illustrated in Figure 7:

1) Systolic-array matrix-multiply (MM) accelerator generated by AutoSA [36].
2) Jacobi3D stencil designs generated by SODA [37].
3) K-nearest neighbors (KNN) in CHIP-KNN [38].
4) Sparse matrix-vector multiplication (SpMV) accelerator generated by Serpens [39].

TABLE I: Summary of Workloads

Workload	FPGA	Memory	# Dice	Most-used
MM [36]	VPK180	DRAM	4	DSP
Jacobi3D [37]	VPK180	DRAM	4	DSP
KNN [38]	VHK158	HBM	2	LUT
SpMV [39]	VHK158	HBM	2	BRAM
Tool Version	AMD Vivado 2024.1			

Table I summarizes the platforms for each workload. We use two types of FPGA boards to evaluate different sizes of the hard NoC and two types of memories to vary the number of memory IOs. For each workload, we create four designs with increasing resource utilization. Table III summarizes the resource utilization of all designs. All designs use at least 40% of one or more types of PL resources. Inter-die crossings also vary from about 6% to over 50%.

Experiment Methodology: To filter the effect of P&R noise, we run five frequency targets—300, 280, 250, 230, and 200 MHz. The maximum frequency results, Fmax, is taken from the five targets. We use Vivado 2024.1, with the default P&R directives.

Baselines: We compare against two baselines: (1) AMD Vivado, and (2) AutoBridge with coarse-grained placement and pipelining (we use the same settings in NoH). We select Vivado to compare in the commercial tool setting. We compare with AutoBridge because NoH also improves frequency using similar coarse-grained placement and pipelining. However, both baselines use the hard NoC only for memory IOs, placing them near memory by default, and do not support automatic NoC utilization for other communication. In contrast, NoH extends coarse-grained placement with hard NoC support and generates offloading configurations. Section VII compares NoH with both baselines in terms of Fmax, simulation cycles, and Super Long Line (SLL) utilization, AMD's metric for inter-die wire usage.

TABLE II: Summary of Frequency and Cycle Count

Configrations	NoH Results		Over Vivado		Over AutoBridge	
	Fmax Results (MHz)	Sim. Cycle Count	Fmax Gain (%)	Cycle Count Overhead (%)	Fmax Gain (%)	Cycle Count Overhead (%)
MM 18x16	182.0	12694	4.2	0.9	*F**	0.43
MM 18x17	190.6	13164	36.6	0.3	*F*	-0.2
MM 18x18	270.5	13742	68.6	0.8	*F*	0.38
MM 18x19	285.8	14671	85.6	1.5	*F*	0.62
J* 109	300.0	2118	19.3	5.7	2.1	4.1
J 115	300.0	2222	30.4	5.5	0.5	3.9
J 121	300.0	2327	8.3	5.3	1.8	3.8
J 124	286.3	2380	17.8	5.1	-4.0	3.7
KNN 27	253.0	11417	9.0	1.8	3.2	1.8
KNN 36	248.8	11654	16.3	0	1.7	1.0
KNN 45	226.6	11681	12.3	1.2	-5.9	-0.2
KNN 54	202.6	11743	0.4	2.9	*F*	1.7
SpMV 32	288.5	3354	7.5	-0.4	5.5	-0.1
SpMV 40	287.3	3458	12.1	1.5	6.4	1.4
SpMV 48	230.0	3606	-6.3	4.5	5.7	3.3
SpMV 56	197.6	3444	-12.0	0.8	46.6	1.6

***Note: J is Jacobi3D. *F* is failed P&R for all 300, 280, 250, 230 and 200 MHz.**

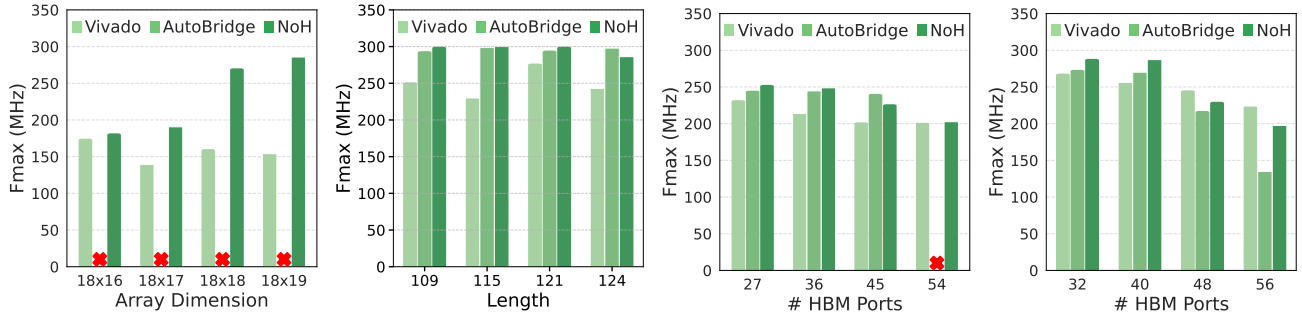
VII. Evaluation

Table II summarizes the overall comparison between NoH and our two baselines. Relative to Vivado, NoH improves Fmax by 20%. Compared to AutoBridge—which failed to route 5 out of 16 configurations, NoH achieves an average Fmax of 226 MHz, with a 6% improvement on the successful configurations. The average cycle count overhead is modest at 2.4% compared to Vivado and 1.7% compared to AutoBridge.

Each workload has been highly optimized for frequency, as noted in their publications. When scaled for Versal FPGAs, the Vivado baseline already averages 217 MHz Fmax, and using five frequency targets helps mitigate noise in P&R results, so any further Fmax improvement is worth noting. We also show that these improvements stem from NoH's techniques for reducing inter-die crossings and congestion.

A. MM

The MM accelerator implements a systolic array. We vary the array size to get up to 77% DSP. The three memory IOs are connected to the same task and placed near the DRAM. This allows NoH to use all NoC bandwidth for the Hard NoC Offloading stage. Its third objective (PL-NoC Cooperative Placement) is exemplified in MM, where a good PL placement



(a) MM (b) Jacobi3D (c) KNN (d) SpMV

Fig. 8: Frequency comparisons of 4 workloads and 16 total design configurations (max. 90% LUT, 97% DSP, and 71% BRAM). A red cross means unroutable. NoH achieves an average of 226 MHz and about 20% frequency improvement over Vivado.

resembles a mesh. Without it, NoC ports could misalign, lowering Fmax.

With hundreds of tasks, AutoBridge struggles to find an optimal solution. It times out after 7 hours while the solver still has over 80% gap, resulting in a placement with more inter-die crossings than Vivado. Consequently, all AutoBridge baselines failed to P&R.

Although NoH uses the same placement, it offloads substantial inter-die crossings to the NoC, reducing the SLL utilization by an average of 19.7% and 37.4% (11172 and 21156 wires) compared to Vivado and AutoBridge, respectively. As seen in Figure 8(a), this leads to a notable Fmax improvement—85% over Vivado. NoH is able to enhance an otherwise failing placement to achieve up to 285 MHz.

B. Jacobi3D

The Jacobi3D accelerator computes multiple iterations of a 7-point 3-dimensional Jacobi computation. We vary the number of iterations to get up to 97% DSP usage. The number of inter-die crossings is small. However, it can demonstrate one ideal use case of hard NoC, where there is enough bandwidth to map all inter-die crossings (excluding scarce control wires). With close to zero crossing, the P&R tool has a very relaxed timing constraint between dice.

Both NoH and AutoBridge have better Fmax than Vivado (Figure 8(b)). Even though the design has used almost all DSP resources, we can still achieve 300 MHz. Our method is slightly worse than AutoBridge for 124-length due to noise in P&R. Nevertheless, our NoH method consistently decreases the SLL utilization by an average of 68.6% from Vivado and 72.7% from AutoBridge (Table III).

C. KNN

The KNN accelerator is LUT-bound. We vary the number of HBM ports to get up to 89% LUT utilization. Figure 8(c) reveals an average improve of 9.5% over Vivado. Vivado has the lowest SLL utilization, but it over-congests the device near the HBM, as seen in Section VII-F. Congestion occurs mainly near HBM controllers due to a large number of memory IOs. NoH allocates all NoC bandwidth for memory IOs to spread

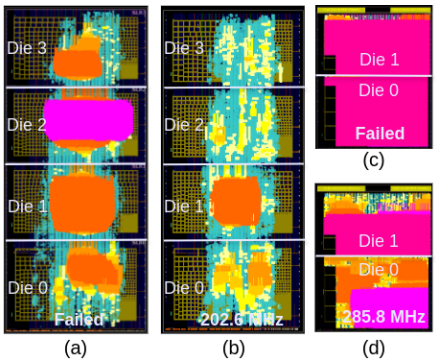


Fig. 9: Congestion comparison: (a, b) MM 18x19 and (c, d) 54-port KNN. Warmer colors show higher congestion; NoH (b, d) achieves lower congestion than AutoBridge (a, b).

out the logic. We consistently reduce inter-die crossings by an average of 28% or 15866 wires compared to AutoBridge.

D. SpMV

The Serpens SpMV accelerator has a more irregular topology. We vary the number of HBM ports to reach up to 71% usage for both BRAM and URAM. SpMV has low inter-die crossings so Vivado can sometimes do better by finer placement granularity. Compared to AutoBridge, on the other hand, NoH consistently reduces SLL utilization, by an average of 20.9% or 11831 wires. We provide an average increase in Fmax of 16.0%, and up to 46.6%. Consistent with the KNN results, NoH proves more effective for designs with more memory IOs, enabling designers to scale up and fully utilize memory bandwidth.

E. Cycle Count

Although the hard NoC is fully buffered in each switch, there is a drop in bandwidth and rise in latency over number of hops. Our ILP algorithm minimizes the effect by using constraints to reserve sufficient NoC bandwidth, virtual channels to isolate independent communication, and the hop-length objective to optimize NoC routes. Table II confirms that the Fmax improvement is not diminished by the overhead,

TABLE III: Resource utilization for all benchmark configurations.

size	MM																		Jacobi3D											
	LUT(%)			FF(%)			BRAM(%)			URAM(%)			DSP(%)			SLL(%)			LUT(%)			FF(%)			BRAM(%)			URAM(%)			DSP(%)			SLL(%)		
	V*	A*	N*	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N
Config1*	11.1	12.4	12.5	3.4	8.6	7.2	37.7	37.7	37.7	0	0	0	64.2	64.2	64.2	52.3	63.5	42.4	13.3	13.5	13.5	28.6	28.8	28.7	16.7	16.7	16.7	0.6	0.6	0.6	85.1	85.1	85.1	5.5	6.5	2.0
Config2	11.7	12.8	12.7	3.7	8.0	6.7	40.0	40.0	40.0	0	0	0	68.2	68.2	68.2	50.0	63.7	41.3	14.0	14.2	14.2	30.1	30.4	30.3	17.6	17.6	17.6	0.6	0.6	0.6	90.0	90.0	90.0	5.9	6.8	1.3
Config3	12.3	13.7	13.8	3.8	9.0	7.5	41.6	41.6	41.6	0	0	0	72.2	72.2	72.2	55.8	71.6	44.4	14.7	15.0	14.9	30.1	32.0	31.9	18.5	18.5	18.5	0.6	0.6	0.6	94.4	94.4	94.4	5.8	6.6	1.9
Config4	13.0	14.4	14.4	4.0	9.0	7.2	43.6	43.6	43.6	0	0	0	76.3	76.3	76.3	54.5	74.2	42.4	15.1	15.3	15.3	32.5	32.7	32.7	19.0	19.0	19.0	0.6	0.6	0.6	96.8	96.8	94.8	5.8	6.6	2.0

size	KNN																		SpMV											
	LUT(%)			FF(%)			BRAM(%)			URAM(%)			DSP(%)			SLL(%)			LUT(%)			FF(%)			BRAM(%)			URAM(%)			DSP(%)			SLL(%)		
	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N	V	A	N
Config1	43.2	42.3	44.0	21.0	22.0	21.8	34	34	34	16.6	16.6	16.6	5.8	5.8	5.8	4.1	8.8	6.0	12.3	12.4	12.3	6.7	8.7	8.3	40.3	40.3	40.3	39.4	39.4	39.4	7.9	7.9	7.9	0	15.5	10.3
Config2	58.1	58.5	58.4	28.0	29.6	29.2	45.3	45.3	45.3	22.1	22.1	22.1	7.8	7.8	7.8	12.9	13.7	9.7	15.2	15.9	15.7	17.8	11.1	10.8	50.4	50.4	50.4	49.2	49.2	49.2	14.3	14.3	14.3	8.8	20.0	14.2
Config3	70.0	74.0	73.2	35.0	37.0	36.6	56.7	56.7	56.7	27.7	27.7	27.7	9.7	9.7	9.7	17.4	18.0	13.6	17.8	18.6	18.1	10.2	12.7	12.4	60.5	60.5	60.5	59.0	59.0	59.0	17.1	17.1	17.1	15.4	21.9	16.7
Config4	89.1	90.2	87.1	42.0	44.4	44.0	68.0	68.0	69.0	33.2	33.2	33.2	11.7	11.7	11.7	9.9	22.6	16.7	21.0	22.7	21.1	11.9	15.7	14.3	70.5	70.5	70.5	68.9	68.9	68.9	19.6	19.6	19.6	17.8	22.9	23.4

***Note: V is Vivado baseline. A is AutoBridge baseline. N is NoH. Config 1 to 4 are in the same order as in Table II for each benchmark.**

with less than 2.3% cycle count overhead compared to both baselines.

F. Congestion Case Study

As shown in the routed device in Figure 9, NoH significantly reduces congestion across all four dies. For MM 18×19, NoH (a) offloads 19,253 inter-die wires, enabling a successful P&R at 286 MHz, while AutoBridge (b) fails. Similarly, for KNN with 54 HBM ports, NoH (c) achieves a balanced placement with fewer wire-detours and inter-die crossings, reaching 203 MHz, whereas AutoBridge (d) again fails to complete P&R.

VIII. Discussion

Scope of HLS Designs: NoH offloads point-to-point, latency-insensitive communication, perfect for task-parallel designs described in TAPA [33] as they provide native abstractions. This approach could be extended to support all-to-all and memory-mapped communications, which are much more costly to implement in PL, but incur no additional cost on the hard NoC. For example, the improvement will be even more notable to offload a soft crossbar.

Applicability to RTL Designs: Our ILP algorithms can operate on RTL-level DFGs provided that latency-insensitive communication is properly annotated. In fact, our current DFG representation is not HLS-specific. Grouping RTL modules into tasks is also important to facilitate manageable ILP runtime.

Other Hard NoC Architecture: Although we tested on the Versal NoC, NoH can be extended to other hard NoC architectures. The P&R constraints for NoC ports, virtual channels, and bandwidth in Section IV-B should be adjusted.

IX. Related Work

FPGA Hard NoC Design and Use: There has been a variety of works exploring FPGA hard NoC micro-architectures and routing [40]–[42]. Several prior works have demonstrated the usefulness of hard NoCs in the context of specific accelerator designs, like Transformers [43], graph convolutional networks [44], and a customizable overlay [45] to reduce compilation time.

Leverage Hard NoCs in Designs: Early works [46], [47] show how FPGA designs can use a hard NoC to save resources and improve frequency before hard NoCs are commercially available. Their works promote hardening NoCs through a handful of case studies, while NoH is the first technique to automatically determine how best to convert designs to use the hard NoC—which latency-insensitive channels to offload and how to offload them. Moreover, we evaluate a variety of real-world designs on multi-die platforms. [48], [49] optimize the computer-aided design (CAD) flow in the open-source VPR framework [50] to support hard NoC, making VPR a viable alternative to commercial tools like Vivado. Future work could integrate NoH into VPR to assess its effectiveness across various NoC architecture and CAD flows.

High-level Physical Synthesis: HLPS works [1], [51]–[54] propose FPGA physical-design techniques to optimize frequency. NoH can be seen as an extension to HLPS tools, like [1], to first consider the hard NoC. HRFF [55] proposes a placement algorithm tailored for a custom NoC-based FPGA architecture using simulators, but it does not provide a generalization of their optimization techniques to other hard-NoC FPGAs.

X. Conclusions

NoH is the first HLS flow that can automatically offload communication to a hard NoC. It formally reasons about resource trade-offs, and enables offloading both memory access and latency-insensitive communication. Our framework cooperatively places PL modules while considering hard NoC communication, using direct modeling for memory IO placement, and a proxy model for placing NoC ports. Our evaluation demonstrates around 20% improvement in frequency over AMD tools by automating hard NoC usage in HLS designs.

XI. Acknowledgments

This work is partially supported by the NSF grant CCF-2200831, the AMD HACC Program, the CDSC industrial partners (https://cdsc.ucla.edu/partners/), and PRISM, one of the seven centers in the JUMP 2.0 program sponsored by SRC and DARPA.

REFERENCES

[1] L. Guo, Y. Chi, J. Wang, J. Lau, W. Qiao, E. Ustun, Z. Zhang, and J. Cong, "Autobridge: Coupling coarse-grained floorplanning and pipelining for high-frequency hls design on multi-die fpgas," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 81–92. [Online]. Available: https://doi.org/10.1145/3431920.3439289

[2] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "Fpga hls today: Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, Aug. 2022. [Online]. Available: https://doi.org/10.1145/3530775

[3] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[4] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 15–24.

[5] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek *et al.*, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 23–32.

[6] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[7] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang, "Fracbnn: Accurate and fpga-efficient binary neural networks with fractional activations," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 171–182.

[8] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[9] 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, 2018.

[10] X. Liu, Y. Chen, T. Nguyen, S. Gurumani, K. Rupnow, and D. Chen, "High level synthesis of complex applications: An h. 264 video decoder," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 224–233.

[11] X. Chen, H. Tan, Y. Chen, B. He, W.-F. Wong, and D. Chen, "Thundergp: Hls-based graph processing framework on fpgas," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 69–80.

[12] G. Dai, Y. Chi, Y. Wang, and H. Yang, "Fpgp: Graph processing framework on fpga a case study of breadth-first search," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 105–110.

[13] J. Zhang and J. Li, "Degree-aware hybrid graph traversal on fpga-hmc platform," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 229–238.

[14] S. Zhou, C. Chelmis, and V. K. Prasanna, "Optimizing memory performance for fpga implementation of pagerank," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2015, pp. 1–6.

[15] A. Parravicini, F. Sgherzi, and M. D. Santambrogio, "A reduced-precision streaming spmv architecture for personalized pagerank on fpga," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 378–383.

[16] S. Zhou, C. Chelmis, and V. K. Prasanna, "Accelerating large-scale single-source shortest path on fpga," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 2015, pp. 129–136.

[17] Y. Chi, L. Guo, and J. Cong, "Accelerating sssp for power-law graphs," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022, pp. 190–200.

[18] J. Zhang and J. Li, "Eniad: A reconfigurable near-data processing architecture for web-scale ai-enriched big data service," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–8.

[19] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martínez, and C. Guestrin, "Graphgen: An fpga framework for vertex-centric graph computation," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2014, pp. 25–28.

[20] T. Oguntebi and K. Olukotun, "Graphops: A dataflow library for graph analytics acceleration," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 111–117.

[21] S. Zhou, R. Kannan, V. K. Prasanna, G. Seetharaman, and Q. Wu, "Hitgraph: High-throughput graph processing framework on fpga," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249–2264, 2019.

[22] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "Graphlily: Accelerating graph linear algebra on hbm-equipped fpgas," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[23] Y.-T. Chen, J. Cong, J. Lei, and P. Wei, "A novel high-throughput acceleration engine for read alignment," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 199–202.

[24] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 127–135.

[25] M. Lo, Z. Fang, J. Wang, P. Zhou, M.-C. F. Chang, and J. Cong, "Algorithm-hardware co-design for bqsr acceleration in genome analysis toolkit," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 157–166.

[26] C. Rauer and N. Finamore, "Accelerating genomics research with opencl and fpgas," *Altera, Now Part of Intel, Tech. Rep*, 2016.

[27] P. Meng, M. Jacobsen, M. Kimura, V. Dergachev, T. Anantharaman, M. Requa, and R. Kastner, "Hardware accelerated novel optical de novo assembly for large-scale genomes," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–8.

[28] L. Wu, D. Bruns-Smith, F. A. Nothaft, Q. Huang, S. Karandikar, J. Le, A. Lin, H. Mao, B. Sweeney, K. Asanović *et al.*, "Fpga accelerated indel realignment in the cloud," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 277–290.

[29] J. Cong, Z. Fang, M. Huang, L. Wang, and D. Wu, "Cpu-fpga co-optimization for big data applications: A case study of in-memory samtool sorting," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 291–291.

[30] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel, "Network-on-chip programmable platform in versal™ acap architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 212–221. [Online]. Available: https://doi.org/10.1145/3289602.3293908

[31] B. Esposito, "Intel agilex® 9 direct rf-series fpgas with integrated 64 gsps data converters," in *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 2023, pp. 1–35.

[32] A. Cairncross, B. Henry, C. Chalmers, D. Reid, J. Shipton, J. Fowler, L. Corrigan, and M. Ashby, "Ai benchmarking on achronix speedster® 7t fpgas," *White Paper*, 2023.

[33] L. Guo, Y. Chi, J. Lau, L. Song, X. Tian, M. Khatti, W. Qiao, J. Wang, E. Ustun, Z. Fang, Z. Zhang, and J. Cong, "Tapa: A scalable task-parallel dataflow programming framework for modern fpgas with co-optimization of hls and physical design," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 4, Dec. 2023. [Online]. Available: https://doi.org/10.1145/3609335

[34] Dally and Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, 1987.

[35] AMD, *Versal ACAP Design Guidance: NoC Simulation*, Advanced Micro Devices, Inc., 2024, accessed: 2024-10-17. [Online]. Available: https://docs.amd.com/r/en-US/ug1273-versal-acap-design/NoC-Simulation

[36] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021.

[37] Y. Chi, J. Cong, P. Wei, and P. Zhou, "Soda: Stencil with optimized dataflow architecture," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE Press, 2018, p. 1–8. [Online]. Available: https://doi.org/10.1145/3240765.3240850

[38] A. Lu, Z. Fang, N. Farahpour, and L. Shannon, "Chip-knn: A configurable and high-performance k-nearest neighbors accelerator on cloud fpgas," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 139–147.

[39] L. Song, Y. Chi, L. Guo, and J. Cong, "Serpens: A high bandwidth memory based accelerator for general-purpose sparse matrix-vector multiplication," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 211–216.

[40] D. U. Becker, "Efficient microarchitecture for network-on-chip routers," Ph.D. dissertation, Stanford University, 2012.

[41] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[42] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1–es, Jun. 2006. [Online]. Available: https://doi.org/10.1145/1132952.1132953

[43] W. Zhang, Y. Liu, and Z. Bao, "Cat: Customized transformer accelerator framework on versal acap," *arXiv preprint arXiv:2409.09689*, 2024.

[44] C. Zhang, T. Geng, A. Guo, J. Tian, M. Herbordt, A. Li, and D. Tao, "H-gcn: A graph convolutional network accelerator on versal acap architecture," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 200–208.

[45] T. Nguyen, Z. Blair, S. Neuendorffer, and J. Wawrzynek, "Spades: A productive design flow for versal programmable logic," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, 2023, pp. 65–71.

[46] M. S. Abdelfattah, A. Bitar, and V. Betz, "Take the highway: Design for embedded nocs on fpgas," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 98–107. [Online]. Available: https://doi.org/10.1145/2684746.2689074

[47] M. S. Abdelfattah and V. Betz, "Lynx: Cad for fpga-based networks-on-chip," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–10.

[48] S. Srinivasan, A. Boutros, F. Mahmoudi, and V. Betz, "Placement optimization for noc-enhanced fpgas," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023, pp. 41–51.

[49] S. G. Shahrouz and V. Betz, "The road less traveled: Congestion-aware noc placement and packet routing for fpgas," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 2024, pp. 33–42.

[50] V. Betz and J. Rose, "Vpr: A new packing, placement and routing tool for fpga research," in *International Workshop on Field Programmable Logic and Applications*. Springer, 1997, pp. 213–222.

[51] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multicycle communication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, 2004.

[52] M. Xu and F. J. Kurdahi, "Layout-driven rtl binding techniques for high-level synthesis using accurate estimators," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 2, no. 4, pp. 312–343, 1997.

[53] T. Alonso, L. Petrica, M. Ruiz, J. Petri-Koenig, Y. Umuroglu, I. Stamelos, E. Koromilas, M. Blott, and K. Vissers, "Elastic-df: Scaling performance of dnn inference in fpga clouds through automatic partitioning," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 2, pp. 1–34, 2021.

[54] L. Du, T. Liang, S. Sinha, Z. Xie, and W. Zhang, "Fado: F loorplan-a ware d irective o ptimization for high-level synthesis designs on multi-die fpgas," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, pp. 15–25.

[55] J. Luo, X. Liu, F. Chen, and Y. Ha, "Hrff: Hierarchical and recursive floorplanning framework for noc-based scalable multidie fpgas," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 11, pp. 4295–4308, 2023.