Placement Optimization for NoC-Enhanced FPGAs

Srivatsan Srinivasan*, Andrew Boutros*, Fatemehsadat Mahmoudi, Vaughn Betz
Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada
{srivatsan.srinivasan, andrew.boutros, sara.mahmoudi}@mail.utoronto.ca, vaughn@eecg.utoronto.ca

Abstract*—Field-programmable gate array (FPGA) architectures have recently incorporated hardened networks-on-chip (NoCs) to enable more efficient and easier system-level integration. However, the embedding of hard NoCs presents a new challenge for FPGA computer-aided design (CAD); the tools need to optimize the placement of circuit netlist primitives to not only minimize total wirelength and critical path delay, but also consider the NoC traffic patterns between modules to minimize their aggregate bandwidth and/or meet latency constraints. This work enables flexible modeling of FPGA architectures with hard NoCs in the open-source versatile place & route (VPR) CAD flow, facilitating both CAD and architecture research. We enhance the placement engine in VPR to co-optimize traditional circuit implementation metrics (e.g. wirelength, critical path delay) and NoC performance metrics (e.g. congestion, bandwidth utilization, latency) when mapping an application design with NoC-attached modules to a candidate NoC-enhanced FPGA architecture. We test our VPR enhancements using a variety of synthetic benchmarks and verify that the placement engine can effectively optimize NoC aggregate bandwidth and meet specified latency constraints. Then, we present a complete flow that integrates VPR with a high-level SystemC architecture simulator, RAD-Sim, that can capture the NoC traffic flows of complete application designs and use it to drive VPR's placement optimizations. We showcase this combined flow using a real application design from the deep learning domain. The results show that our NoC-enhanced VPR flow can result in 2× reduction in NoC aggregate bandwidth (on average) compared to a NoC-agnostic flow, without affecting the design's wirelength or critical path delay.

I. Introduction

Modern field-programmable gate arrays (FPGAs) contain millions of programmable logic elements, along with thousands of block RAMs (BRAMs), digital signal processing (DSP) blocks, high speed I/Os, and more [1]. This enables the implementation of large complex FPGA systems, and to maintain design productivity, these systems typically consist of many intellectual property modules (IPs). The FPGA fine-grained programmable routing provides the flexibility to implement any custom connectivity within an IP, as well as the larger-scale connectivity between IPs. While bit-level programmable routing is well suited to customized, local communication, it can be challenging to use it to implement efficient *system-level interconnect* which links multiple IPs.

Closing timing on such cross-system links is becoming more challenging as process technology advances provide more logic density but increased long-distance wire delays [2]. For example, a recent study showed that the number of interconnect pipeline stages required to run a 32-bit bus connecting two IPs at opposite corners of a mid-size FPGA at 400 MHz has increased from 3 stages in a 20 nm Arria 10 device to 10 stages in a 14 nm Stratix 10 device [3]. The difficulty of designing system-level interconnect is further exacerbated by the ever-increasing bandwidth of memory and I/O interfaces in FPGAs. For example, many recent FPGAs have multiple high-bandwidth memory (HBM) stacks, 100 Gbps Ethernet interfaces, as well as several PCIe and DDR memory controllers [4]. To distribute this tremendous bandwidth to different design IPs on the FPGA fabric, the *soft* (i.e. programmable) FPGA routing must implement very wide buses, consuming significant resources and leading to further timing closure challenges. Developing a bespoke system-level interconnect for each application not only costs resources but also slows time-to-market by adding a complex integration task and increasing design iterations to close timing.

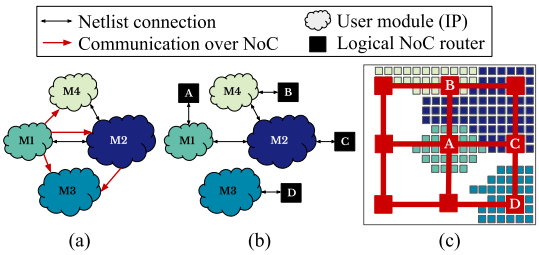

Fig. 1: Illustration of the NoC placement problem: (a) logical view of the system, (b) circuit netlist with logical NoC router instantiations, and (c) placement of logical NoC routers and user modules to physical NoC routers and FPGA resources.

To ease these system-level interconnection challenges, FPGAs have begun embedding *hard* packet-switched networks-on-chip (NoCs). A hardened mesh NoC with 16 routers and 128-bit links would cost less than 2% of the FPGA die area and provide the same bandwidth of a conventional soft bus using 14% of the FPGA resources while consuming an order of magnitude less power [5]. In addition, a hard NoC completely decouples computation from communication enabling easier and faster integration of various IPs in a complex system. Designers can just focus on optimizing independent IPs to meet timing and connect them to the NoC for inter-IP communication, which guarantees system-level interconnect timing closure. Due to the substantial gains they offer at a relatively low silicon cost, embedded hard NoCs were recently introduced in commercial FPGAs from AMD [6], Achronix [7], and Intel [4].

However, embedding a hard NoC in the FPGA fabric comes with two major challenges. Firstly, the reconfigurable nature of an FPGA means that the performance requirements and use cases of the NoC are unknown when architecting the device. Therefore, FPGA architects have to select NoC specifications (e.g. number of routers, network topology, link bandwidth, number of virtual channels) that can achieve efficient system-level communication across a wide variety of applications to maintain the FPGA generality and flexibility. But since most FPGA application designs are developed for conventional FPGAs, an application-architecture co-design framework is necessary for exploring the design space of NoC-enhanced reconfigurable devices. Secondly, the FPGA physical design

*Both authors contributed equally to this work.

tools must also be NoC-aware; while optimizing the conventional circuit netlist placement to minimize critical path delay (CPD), the FPGA CAD tools need to simultaneously optimize the placement of NoC-connected modules to minimize NoC congestion, bandwidth utilization and/or latency. We refer to this as the *NoC placement* problem, which is our main focus.

An illustration of the NoC placement problem is shown in Fig. 1. To connect modules to the embedded NoC, the designer would instantiate *logical routers* in their design and connect them to the corresponding module interfaces. The NoC placement is the mapping of these logical routers to *physical router* locations on the FPGA. This is analogous to mapping circuit netlist primitives to logic block, BRAM, and DSP locations in conventional FPGA placement to minimize the delays of the netlist connections between these primitives. However, the inter-module communication patterns over the NoC cannot be captured as connections in the circuit netlist as they depend on the runtime behavior of the user application. Therefore, the *NoC traffic flows* (i.e. communication between logical routers over the NoC) along with their bandwidth requirements and optional latency constraints must also be input to the CAD tool to optimize NoC placement. These traffic flows can be directly specified by the designer based on their knowledge of the application; however, this can be challenging especially for large and complex application designs. Another alternative is to use trace-based traffic flows extracted from end-to-end application simulation. This would benefit from a fast high-level simulator for NoC-enhanced FPGAs to avoid the long runtime of register transfer level (RTL) simulations of the whole system. Such a simulator could also potentially evaluate the quality of different NoC placement solutions proposed by the physical design tools in intermediate optimization steps.

In this work, we first enhance the open-source versatile place & route (VPR) tool [8] to enable modelling of hard embedded NoCs in a given FPGA architecture*. We also specify a new input file format for describing NoC traffic flows and develop new steps in the CAD flow for routing these traffic flows over the NoC and optimizing the NoC placement. We test our new NoC-enhanced VPR flow using a variety of synthetic benchmarks to show that the new flow can indeed optimize the NoC placement and minimize the utilized aggregate NoC bandwidth. Finally, we showcase our NoC placement approach using a real application design from the deep learning (DL) domain. We extract trace-based traffic flows of the application using RAD-Sim [9], a SystemC architecture simulator for NoC-enhanced reconfigurable acceleration devices (RADs), to drive the NoC placement optimization and then evaluate the proposed placement solutions from VPR. To the best of our knowledge, this work is the first to:

- support modelling of hard embedded NoCs in an open-source FPGA CAD flow to enable architecture exploration of NoC-enhanced FPGA architectures,
- integrate NoC placement optimization in VPR and showcase its utility using a variety of synthetic benchmarks, and
- use the combination of an application-driven architecture simulator and FPGA physical design tools (RAD-Sim and VPR) for automated trace-based NoC placement.

*Our enhancements are integrated in the Verilog-to-Routing master branch at https://github.com/verilog-to-routing/vtr-verilog-to-routing

II. Background & Related Work

A. FPGA Packet-Switched NoCs

It is becoming increasingly challenging to distribute the ever-growing transceiver and external memory bandwidth to various application modules in modern FPGA systems. These external interfaces typically run at significantly higher clock speeds than the FPGA programmable logic and routing, and therefore require wide deeply-pipelined buses on the FPGA fabric (i.e. *soft buses*) to match their data rates. For example, a Xilinx Ultrascale+ device comes with two HBM stacks, each of which has 16 independent 64-bit channels running at 900 MHz double data rate [10]. This tremendous HBM bandwidth can only be matched using an 8192-bit soft bus running at a relatively high 450 MHz clock frequency. Such system-level interconnect can rapidly use a major fraction of the FPGA logic and routing resources, especially when spanning large distances and requiring deep pipelining to meet timing. Furthermore, the HBM hardened memory controllers are typically located on one side of the chip (to facilitate the implementation of soft or hard inter-channel crossbar switch [11]) resulting in severe routing hot spots.

As an alternative, packet-switched NoCs have been extensively studied as a more efficient system-level interconnect for FPGAs. Soft NoCs with routers and links implemented using the FPGA's programmable logic and routing were explored as a solution that can be implemented on off-the-shelf conventional FPGAs. Starting from the system-level interconnect requirements, an expert designer would manually design the soft NoC and pre-compile it as an interconnect *overlay* that can be directly instantiated to connect user design modules [12], [13]. An FPGA-specific soft NoC generator was also introduced in [14] to automate the NoC design process and democratize the use of NoCs in modern FPGA designs. Although soft NoCs can facilitate timing closure of system-level interconnect, they still consume a considerable portion of the valuable programmable FPGA resources [15].

Abdelfattah and Betz evaluated the area cost and efficiency gains of the alternative approach of embedding a hard packet-switched NoC into the FPGA fabric [16]. In this case, the NoC routers are implemented using ASIC standard cells (similar to other FPGA hard blocks) with dedicated links between them and *fabric ports* to interface with the programmable routing and implement width adaptation, clock domain crossing as well as any necessary arbitration. It was shown that a full-featured, hard mesh NoC is 23× more area efficient, 6× faster, and consumes 11× less power than its soft implementation [5]. This comes at the cost of less than 2% of the FPGA die size for a 4×4 mesh NoC with 128-bit links. In [17], the authors also showed that hard NoCs can be very well suited for virtualized datacenter FPGAs; they can significantly improve resource utilization, operating frequency and routing congestion of the FPGA *shell* connecting one or multiple user design *roles* to various external interfaces.

More recently, FPGA vendors have incorporated hard NoCs in their latest commercial architectures. The Xilinx Versal device embeds a hard NoC that provides standard AXI interfaces to connect user modules on the FPGA fabric to different system components such as the software-programmable ARM cores, AI vector processors, and high-speed transceivers [18]. The NoC has 128-bit links running at 1 GHz (to match a

DDR3 channel bandwidth) and is organized in a *squished mesh* topology where the horizontal links are pushed to the top and bottom of the device to minimize the disruption to the FPGA's column-based layout [6]. The Achronix Speedster7t device also has a hard NoC that consists of a high-bandwidth peripheral ring topology connected to many high-bandwidth external interfaces (e.g. Gen5×16 PCIe, 4×400 Gbps Ethernet interfaces, 8×GDDR6 interfaces), and independent horizontal/vertical NoCs to distribute this bandwidth throughout the programmable fabric [7]. The recent Intel Agilex-M device incorporates two hard NoCs at the top and bottom of the device to distribute the bandwidth of two HBM2e stacks and several off-chip memories to user modules on the FPGA fabric [4]. These NoCs allow users to efficiently implement designs with a fully hardened crossbar, where any on-fabric initiator can read/write data to any target memory interface.

In this work, we focus on the implications of incorporating hard packet-switched NoCs on the FPGA computer-aided design (CAD) tools, and more specifically the placement of a design netlist on the FPGA fabric while considering inter-module communication over the NoC. However, our approach is not fundamentally limited to hard NoCs and can be applied to the placement of designs that include a pre-compiled soft NoC overlay as a system-level interconnect solution.

B. NoC Simulators

Almost every modern many-core CPU and GPU chip, as well as many custom application-specific accelerators, use some form of an NoC as a system-level interconnect solution. Booksim [19] is a C++ open-source NoC simulator commonly used to evaluate the performance of different NoC architectures, micro-architecture implementations, and routing policies under different synthetic statistically generated traffic patterns. It was leveraged by many architecture simulators (e.g. GPGPU-Sim [20] and SIAM [21]) to model the overall system performance including NoCs. Unlike CPU, GPU and custom (ASIC) accelerator architectures, the design of FPGA NoCs is a more challenging task since both the functionality and placement of NoC-attached processing elements are determined only at compile time due to the FPGA's reconfigurability. FPGA architects have to design the system-level NoC to be suitable for various key FPGA use cases without compromising flexibility.

Several CAD tools were introduced to aid the design of FPGA embedded NoCs. RTL2Booksim [22] allows the co-simulation of FPGA RTL modules and a C++ Booksim NoC model. It was used to evaluate the performance of several application designs implemented on a NoC-enhanced FPGA. However, this approach suffered from slow turnaround time due to the long runtime of RTL simulation. It also required implementing all application modules in a low-level hardware description language (HDL), which can be a labor-intensive process, especially at early design stages when both the application and the NoC architectures are being co-optimized.

More recently, Boutros et al. introduced RAD-Sim, an application-driven simulator that can be used for architecture exploration of novel reconfigurable devices that incorporate an FPGA fabric, a system-level NoC, and coarse-grained accelerator blocks [23]. A high-level SystemC description of application modules along with RAD architecture specifications and NoC placement constraints are given as inputs to RAD-Sim, which performs rapid cycle-accurate simulation and produces end-to-end application performance and NoC traffic reports. They presented a case study using a DL inference overlay to demonstrate the utility of RAD-Sim in rapidly co-optimizing application and device architecture as well as modeling the complex interactions between different system parameters for a wide variety of monolithic, 2.5D and 3D integrated RADs [9]. In this work, we use RAD-Sim to simulate an application design and output its NoC traffic flows to drive the NoC placement optimization in VPR.

C. The NoC Mapping & Placement Problem

Prior work has investigated the problem of mapping compute cores (in a system-on-chip) onto a given NoC architecture to optimize performance and energy, or ensure deadlock-free operation [24]–[27]. This is performed once at the design phase and then the optimized organization of the chip is physically implemented by ASIC design tools. Due to the hardware reconfigurability of FPGAs, the NoC mapping and placement have to be performed during bitstream compilation for each new design implemented on the FPGA. The FPGA resource utilization and programmable routing congestion along with the NoC traffic patterns and latency/bandwidth constraints all have to be considered when optimizing the NoC placement as well as the placement of the circuit netlist primitives. In [28], the authors introduced `LYNX`, a CAD tool that can automatically connect FPGA application modules using an embedded NoC. Starting from a user-specified application connectivity graph and NoC architecture specification, it clusters the application modules, decides which connections should be mapped to the NoC vs. programmable routing, maps the clustered modules to NoC routers, and finally generates RTL wrappers for the user to drop in their application modules. However, `LYNX` uses an analytical cost function and simulated annealing to perform NoC mapping, without consideration of the dynamic changes in the communication patterns of the application or the interaction between the placement of circuit netlist primitives and the suggested NoC mapping.

AMD provides a NoC compiler that performs the NoC placement and routing for the Versal device family as part of their Vitis toolflow [6], [29]. The NoC compiler takes a description of NoC traffic flows (source, destination, bandwidth requirement, priority) as input from the user. It then places the endpoints of traffic flows to NoC interfaces and routes between them. Finally, it evaluates the produced solution and informs the user whether the specified constraints are met or not. While the details of the Vitis NoC compiler are not published, its user guide [30] mentions that it can be invoked by the placement engine to generate a new NoC placement if the initial one causes legality issues or appears to be a poor match to the global placement of the programmable logic. In contrast, our work integrates the NoC placement optimization into VPR's detailed placement engine, allowing simultaneous co-optimization of the NoC mapping and placement of conventional FPGA resources. The placement algorithm optimizes the mapping of logical routers to physical router locations similarly to any other netlist primitive. As well, AMD Vitis is closed-source and targets specific AMD devices, precluding use by FPGA architecture researchers and those wishing to implement new CAD algorithms.

III. Our NoC Placement Flow

In this section, we present an overview of our NoC placement flow that combines RAD-Sim for application-driven simulation with a NoC-enhanced version of VPR for physical design. We will also introduce the implementation details of our VPR enhancements to support NoC modeling, traffic flow description, and NoC placement optimization.

A. Definitions

To precisely detail our flow, we first define the following terms:

- A **logical router** represents a *black-box* NoC router module instantiated in the user design netlist and connected to an application module that requires access to the NoC.
- A **physical router** is the actual hard block NoC router embedded in the FPGA fabric.
- A **link** is the set of wires connecting two physical routers.
- The **NoC topology** is the organization of physical routers and the links between them to form an on-chip communication network (e.g. mesh, butterfly, fat tree).
- A **flit** is the smallest quantum of data traversing the NoC. One or more flits constitute a **packet** which represents a complete *message* transferred from a **source** module to a **destination** module over the NoC.
- The **unloaded router latency** is the best-case delay for a flit to traverse through a physical router, i.e. when there is no queuing delay.
- The **link latency** is the delay for a flit to traverse from one physical router to the next through a NoC link.
- A **connection** (s, d) is a one-way communication from a source logical router s to a destination logical router d.
- The **connection bandwidth** is the application-dependent data transfer rate for a connection in bits per second (bps).
- The **connection latency constraint** is the maximum allowable delay for data transfers in a given connection.
- The **connection priority** is the importance/criticality of a connection relative to other connections in the user design.
- A **traffic flow** is the encapsulation of a connection and its characteristics (bandwidth, latency constraint, priority).
- The **traffic flow aggregate bandwidth** is the total link bandwidth utilized by a traffic flow after its source and destination logical routers are mapped to physical routers, and the connection between them is routed over the NoC (i.e. one or more physical routers and links are traversed to get from source to destination). It can be calculated as:

$$BW_{agg}(T) = N_{links} \times BW(T) \quad (1)$$

  where N_{links} is the number of links in the routed path and $BW(T)$ is the connection bandwidth of traffic flow T.
- The **traffic flow latency** is the total time of a traffic flow data transfer after its source and destination logical routers are mapped to physical routers, and the connection between them is routed over the NoC. It can be calculated as:

$$Lat(T) = N_{links} \times L_{link} + N_{routers} \times L_{router} \quad (2)$$

  where N_{links} and $N_{routers}$ are the numbers of traversed links and routers on the routed path, L_{link} is the link latency, and L_{router} is the unloaded router latency. Congestion in the NoC could increase the traffic flow latency above this value, but during placement optimization, we are guided by this lower bound.

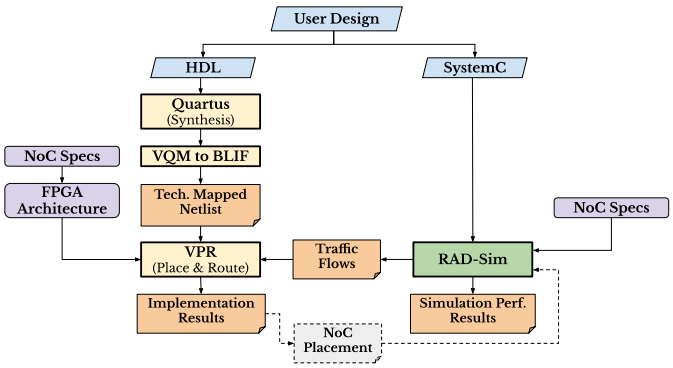

Fig. 2: Our proposed flow combines NoC-enhanced VPR and application-driven simulation using RAD-Sim. VPR co-optimizes the placement of circuit netlist primitives and NoC logical routers driven by application NoC traffic flows generated using RAD-Sim.

B. Flow Overview

Fig. 2 shows an overview of our proposed NoC placement flow. In the first component of the flow, RAD-Sim takes as inputs a SystemC description of the user design modules along with user-defined NoC specifications. It performs end-to-end simulation assuming a random assignment of user design modules to NoC routers (i.e. random NoC placement) to generate the application traffic flows. On the other hand, the HDL implementation of the complete user design (including black-box instantiations of logical routers connected to design modules that require access to the NoC as illustrated in Fig. 1) is synthesized using the Intel Quartus Prime tool. The synthesis and technology mapping steps can be alternatively performed using one of the Verilog-to-Routing (VTR) open-source front-ends (e.g. ODIN-II [31] or Yosys [32]). However, in this work, we use the Intel Quartus synthesis front-end due to its better language coverage and quality of results.

Then, we use the VQM2BLIF tool from the Titan flow [33] to convert the technology-mapped netlist generated by Quartus as a Verilog Quartus Mapping (VQM) file into the Berkeley Logic Interchange Format (BLIF) consumed by VPR. We also extend the `XML` VPR architecture description file with new tags to allow the description and modeling of embedded NoCs. The FPGA architecture description file, the BLIF netlist of the user design, and the application NoC traffic flows generated by RAD-Sim are then passed as inputs to VPR. During the placement stage, our NoC-enhanced VPR optimizes the NoC placement to minimize the NoC aggregate bandwidth and latency while simultaneously considering the effect of NoC placement on the placement of the rest of the circuit netlist primitives and the programmable routing connections between them. Finally, VPR produces the FPGA implementation results (resource utilization and maximum operating frequency) as well as its NoC placement solution. This NoC placement can be also fed back to RAD-Sim to evaluate its effect on end-to-end application performance compared to the initial random NoC placement.

Our proposed approach integrates the NoC placement optimization into the conventional placement step of the FPGA CAD tools. This gives the placement optimization engine full flexibility when mapping logical routers to physical router locations, similarly to any other circuit netlist primitive, to co-optimize both circuit performance metrics (e.g. timing,

wirelength) and NoC performance metrics (e.g. aggregate bandwidth, latency). Our approach not only allows the user to manually specify traffic flows and arbitrary NoC latency constraints if desired (similar to the AMD Vitis NoC compiler), but also enables the automatic generation of application-specific traffic flows from RAD-Sim to drive the NoC placement optimization. This requires SystemC and HDL implementations of the same user design modules to drive both RAD-Sim and the physical design tools, respectively. However, this can be avoided by either generating HDL from SystemC using a high-level synthesis tool such as Catapult HLS [34], or extending RAD-Sim to perform HDL/SystemC co-simulation at the cost of longer runtime.

C. VPR Enhancements for NoC Modeling and Optimizations

In this work, we enhance both the front-end and the core placement engine in VPR to enable future research on FPGA architecture and CAD for devices with hard embedded NoCs. To maintain the generality of VPR, we first extend its XML-based FPGA architecture description syntax to enable users to flexibly model hard NoCs with different specifications. Then, we introduce a new VPR input file that specifies the traffic flows between logical routers to drive the NoC placement optimizations. Finally, we modify the VPR placement engine to co-optimize for NoC-related metrics such as latency and aggregate bandwidth. In this sub-section, we cover the implementation details of these three main enhancements.

1) NoC Description in VPR Architecture File: The first step in modeling hard NoCs in VPR is to allow users to flexibly describe the NoC specifications as part of the FPGA architecture they are experimenting with. Listing 1 is a snippet from an architecture description file that shows how users can now specify an embedded hard NoC in their architecture. The `<noc>` tag provides high-level NoC specifications such as the link bandwidth, link latency, and the physical router tile/block name. Then, the subtag `<topology>` is used to describe the general NoC organization. Each entry in this subtag specifies the identifier (`id`) and XY grid locations (`positionx, positiony`) of a physical router, as well as the IDs of the physical routers to which it is connected (`connections`). This allows the specification of any desired custom NoC topology. The example in Listing 1 describes a simple 2×2 regular mesh NoC with routers at grid locations (0, 0), (0, 5), (5, 0) and (5, 5), link bandwidth of 120 Gbps and link/router latencies of 1 ns. The NoC description is parsed from the architecture file and stored in internal VPR data structures to be used by the placement optimization engine. We also enhance the VPR graphical user interface to visualize the specified hard NoC as shown in Fig. 3 for an example AMD-Versal-like topology.

2) NoC Traffic Flows Input File: To be able to co-optimize for NoC-related metrics (e.g. latency, aggregate bandwidth) during placement, the CAD tool needs a description of the NoC traffic flows in the design. Therefore, we introduce a new XML-based input file to VPR which specifies the traffic flows between *logical* NoC routers. This file can be manually written by the user based on their knowledge of the application design (as in our experiments with synthetic designs in Section IV) or automatically generated by simulating the application design (as in our case study using RAD-Sim in Section V). Listing 2 shows an example traffic flow input file.

```
<!-- Description of a 2x2 mesh NoC-->
<noc link_bandwidth="1.2e9" router_latency="1e-9"
     link_latency="1e-9"
     noc_router_tile_name="noc_router_adapter">
   <topology>
      <router id="0" positionx="0" positiony="0"
          connections="1 2"/>
      <router id="1" positionx="5" positiony="0"
          connections="0 3"/>
      <router id="2" positionx="0" positiony="5"
          connections="0 3"/>
      <router id="3" positionx="5" positiony="5"
          connections="1 2"/>
   </topology>
</noc>
```

Listing 1: A snippet from a VPR architecture description file specifying a simple 2×2 mesh NoC.

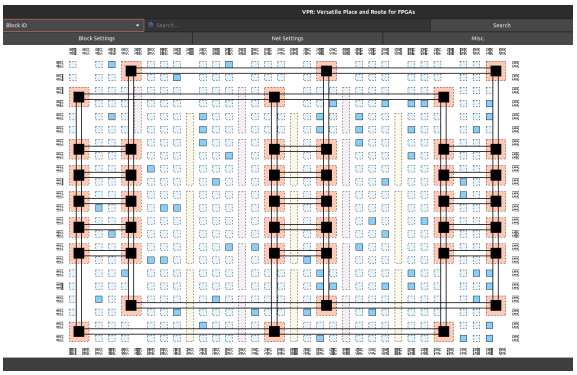

Fig. 3: A screenshot from the VPR GUI showing a device with a custom AMD-Versal-like hard NoC topology. The routers and links of the NoC are shown as black squares and lines, respectively.

The `<single_flow>` tag specifies the source/destination *logical* routers and the bandwidth of this flow. The traffic flow description can also optionally include a latency constraint (analogous to a conventional timing constraint) and/or a flow priority which specifies the weighting factor of this flow in the NoC placement optimization cost function. If a latency constraint is not specified, the NoC placement optimization tries to minimize the latency of the traffic flow as much as possible, and by default, all traffic flow priorities are set to 1.

3) NoC Placement Optimization: The standard VPR placement engine starts with random initial placement and then uses simulated annealing (SA) to optimize this initial solution [8]. In SA, a random swap or a directed move [35] is proposed for one or more netlist primitives, and a certain cost function is calculated for the new placement. If the cost is improved, the *placement perturbation* is accepted and if the cost is degraded, it can be randomly accepted with a probability that decreases throughout the placement iterations (based on the *annealing temperature*). This allows the placement optimization to escape local minima by sometimes accepting worse solutions, and also gradually reduces the probability of accepting worse solutions as the optimization progresses toward a more stable solution. The cost function of the standard VPR placement engine ($C_{netlist}$) focuses on minimizing conventional circuit implementation metrics such as the wirelength and CPD and would not consider the effect of placement perturbations on NoC performance metrics. In other words, perturbing the assignment of logical routers to physical routers using the standard VPR placer would result in random changes to

```
<traffic_flows>
 <single_flow src="m0" dst="m1" bandwidth="2.3e9"
       latency_cons="3e-9"/>
 <single_flow src="m0" dst="m2" bandwidth="5e8"/>
 <single_flow src="ddr" dst="m0" bandwidth="1.3e8
      " priority=3/>
 <single_flow src="m3" dst="m2" bandwidth="4.8e9"
       latency_cons="5e-9" priority=2/>
</traffic_flows>
```

Listing 2: An example description of application design traffic flows.

the NoC performance, as the optimization engine would not evaluate if one NoC placement solution is better or worse than another. Therefore, we modify the standard VPR placer to co-optimize both conventional as well as NoC performance metrics by introducing two new components to the SA cost function:

- **NoC Aggregate Bandwidth**: The sum of the aggregate bandwidths of all traffic flows. Minimizing this metric reduces NoC bandwidth utilization and congestion by avoiding long travel paths for traffic flows (i.e. fewer physical NoC routers and links are traversed by a traffic flow).
- **NoC Latency**: The sum of the unloaded latencies of all traffic flows. Minimizing this metric reduces the NoC communication latency between application modules and helps meet any maximum latency constraints specified.

Eq. 3 and 4 are the two new cost function components for NoC aggregate bandwidth (C_{bw}) and latency (C_{lat}). $BW_{agg}(T_i)$ and $Lat(T_i)$ are the aggregate bandwidth and latency of the traffic flow T_i as defined in Eq. 1 and 2, $P(T_i)$ and $LatConst(T_i)$ are the priority and maximum latency constraint of the traffic flow T_i from the traffic flows input file, and α and β are empirically determined hyper-parameters to specify different weights for the two terms of C_{lat}. Although not shown in the formulae, these two components have different units and value ranges (e.g. $C_{bw} \sim 10^9$ bps and $C_{lat} \sim 10^{-9}$ sec). Therefore, they are first normalized to equalize their magnitudes and then added to the conventional placement cost function ($C_{netlist}$) to calculate the new SA cost function as shown in Eq. 5. Another hyper-parameter, ω, is used to control the weight of the NoC-related cost components relative to the conventional ones.

$$C_{bw} = \sum_{T_i \in \mathbb{T}} P(T_i) \times BW_{agg}(T_i) \tag{3}$$

$$C_{lat} = \sum_{T_i \in \mathbb{T}} P(T_i) \times \Big(\alpha \times \mathtt{max}\big(0, Lat(T_i) - LatConst(T_i)\big) + \beta \times Lat(T_i) \Big) \tag{4}$$

$$C_{Total} = C_{netlist} + \omega \times (C_{bw} + C_{lat}) \tag{5}$$

We modify the SA algorithm to update the additional NoC-related cost components only when the netlist primitives moved are logical routers. For each logical router moved, we iterate over all the traffic flows with this router as the source or destination, and re-route these flows to calculate their new aggregate bandwidth and latency. We implement two common NoC routing algorithms: dimension-ordered (XY) routing for the 2D mesh topology [36] and minimal breadth-first search routing for any other NoC topology. Our NoC placement optimization is generic and the code is written so that a new NoC routing algorithm (which must be deterministic) can be added by writing an implementation of the abstract `NoCRouting` class.

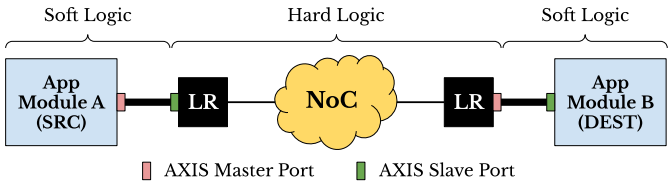

Fig. 4: Modules connected to black-box logical router (LR) IPs as access points to the NoC. Our modified Titan flow pushes the design through Quartus synthesis & generates a BLIF netlist for VPR.

IV. Flow Evaluation using Synthetic Designs

In this section, we test our VPR enhancements for NoC placement using a set of synthetic designs and manually written NoC traffic flows. These test cases can be considered micro-benchmarks for which we know the expected outcome to verify that the enhanced VPR placement engine is behaving as intended. We first explain how a design is prepared and pushed through our NoC-enhanced VPR, and then we describe the set of synthetic benchmarks that we experiment with. Finally, we present the results of our NoC placement flow to show that they match the desired outcomes for these designs.

A. Design Preparation

To implement designs with NoC-attached logic, we implement a black-box logical router Verilog module that can be instantiated and connected to modules that need to access the NoC. Our logical router IP presents a standard master/slave AXI-streaming (AXIS) interface to the application modules as illustrated in Fig. 4. The logical router module internally consists of the NoC router along with a NoC adapter that is responsible for packetizing an AXIS transaction into NoC flits and vice versa, as well as performing any required frequency/width adaptation similar to that in [9]. From an implementation standpoint, the NoC adapter can either be hardened as part of the physical hard router or mapped to the FPGA's soft logic. Studying the trade-off between a hard, soft, or mixed NoC adapter is tangential to this work and, for simplicity, we assume a hardened NoC adapter (i.e. a logical router is directly mapped to a physical router without any additional soft logic).

To prepare a design for our NoC-enhanced CAD flow, we write the design top-level to instantiate all the design modules and connect them to each other and to logical router modules when they access the NoC. We also modify the Titan flow [33] to support custom hard blocks. It can now automatically set user-specified modules that should be implemented with new hard blocks (logical routers in our case) as empty design partitions, and adds synthesis directives to prevent the tool from optimizing away the input/output port registers of these modules during synthesis. Then, it pushes the design through the Quartus synthesis engine and uses our modified VQM2BLIF tool to convert the Quartus synthesized netlist to a BLIF netlist with logical routers instantiated as hard block netlist primitives to be later mapped by VPR to physical router blocks described in the FPGA architecture XML file.

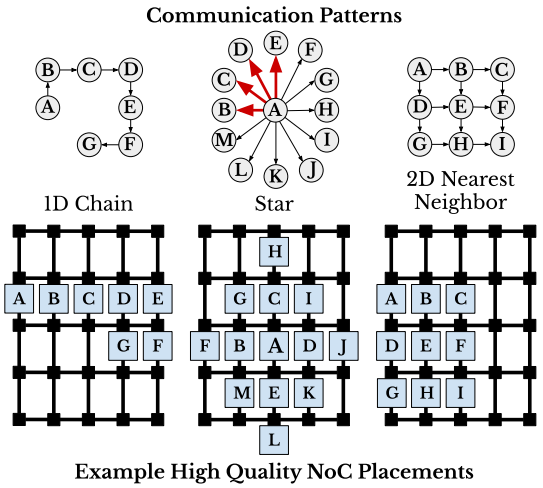

Fig. 5: Communication patterns and example high quality NoC placements for our 1D chain, star, and 2D nearest neighbor benchmarks.

B. Synthetic Benchmarks

To test our NoC placement, we create a set of synthetic benchmarks that have different NoC traffic patterns, bandwidth requirements, latency constraints, and the number and size of application modules. These benchmarks are designed such that high-quality (sometimes optimal) placements for them are known, allowing us to evaluate the achieved NoC placement quality against these known solutions; similar techniques have been used to evaluate ASIC placement tool quality [37]. Fig. 5 depicts the three different types of synthetic benchmarks we implement as well as an example of expected high-quality placement for each of them. Our synthetic benchmarks can be described as follows:

1) 1D Chain: This benchmark contains NoC-attached modules with traffic flows going from one module to the next in a serial 1D chain ($A \rightarrow B \rightarrow C \rightarrow ...$). The traffic flows between logical routers have tight latency constraints that can be met only if they span a single NoC link. An optimal solution would place communicating logical routers at neighboring physical router locations in a snake-like layout.

2) Star: This benchmark creates a 1-to-N scatter communication pattern where one application module communicates with all other modules in the design ($A \rightarrow \{B, C, D, ...\}$). We implement two variations of this benchmark: one with equal bandwidth requirements and no latency constraints for all traffic flows, and another with gradually decreasing bandwidth requirements and relaxing latency constraints for different traffic flows. For both variations, high-quality solutions place the single source logical router at a central physical router with all other destination logical routers placed around it. Additionally, for the second variation, we expect the destination logical routers with higher bandwidth and tighter latency traffic flows to be placed closer to the central source router (as shown by the red bold traffic flows from A to B, C, D, E in Fig. 5).

3) 2D Nearest Neighbor: The last type of synthetic benchmarks implements a 2D grid of application modules with one direction nearest neighbor traffic flows ($A \rightarrow \{B, D\}, B \rightarrow \{C, E\}, ...$) as shown in Fig. 5. Similarly to the 1D chain benchmark, we set the latency constraints of the traffic flows to be satisfied only when the communicating logical routers are mapped to neighboring physical routers.

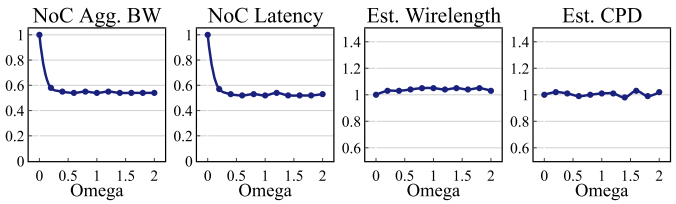

Fig. 6: Effect of the value of ω on different metrics. $\omega = 0.6$ achieves most of the benefits for NoC aggregate bandwidth and latency with almost no effect on post-placement estimated wirelength and CPD.

For each of these benchmark types, we implement two variations (which we refer to as *simple* and *complex* for the rest of this section). For the simple variations, the application modules connected to logical routers are simple 32-bit counters and accumulators (i.e. tens of netlist primitives). On the other hand, the complex variations have relatively larger and more complex application modules implementing FIR filters and SHA encryption cores (i.e. hundreds to thousands of netlist primitives). For the results in this section, we use the naming convention `<s/c>-<type>-<R>r-<X>bw/<Y>lat` to specify the simple/complex (s/c) variation, the benchmark type (as 1D chain, star, or 2D nearest neighbors), number of logical routers (`R`), and the number of different bandwidth requirements (`X`) or latency constraints (`Y`) for the travel flows. For example, `c-nn2d-64r-112lat` identifies the complex variation of the 2D nearest neighbor benchmark with 64 logical routers and 112 latency constraints.

C. Experimental Results

We run all our experiments on an Intel Xeon Gold 6146 CPU with a 3.2 GHz clock speed and 768 GB of RAM. Our NoC-enhanced VPR builds on top of the latest VPR 8 version from the Verilog-to-Routing Github repository [38]. All the results presented in the rest of this paper are averaged across 5 seeds to minimize the effect of CAD noise and run variations.

1) Tuning Hyper-parameters: We run experiments to empirically select the placement cost function hyper-parameters (α, β, ω) in Eq. 4 and 5. We first set both α and β values to 1, and sweep the value of ω for a suite of our simple and complex synthetic benchmarks. Fig. 6 shows the geometric average of the NoC metrics (aggregate bandwidth and latency) as well as post-placement estimates of wirelength and CPD relative to running with NoC placement optimization disabled ($\omega = 0$). The results show that increasing the value of ω beyond 0.6 does not provide any additional benefits for the NoC aggregate bandwidth and latency. On the other hand, the estimated wirelength and CPD slightly increase when $\omega > 0$, but are almost unaffected by the value of ω. Therefore, we pick $\omega = 0.6$, which achieves good NoC performance without negatively affecting the optimization of conventional metrics.

Fig. 7 shows the percentage of the user-specified latency constraints satisfied by the placement engine when $\alpha = 0$ and $\alpha = 1$ (i.e. the effect of adding the latency constraint term to the cost function in Eq. 4). The results show that when α is set to 1, the tool successfully satisfies almost all the latency constraints. Note that in some of our benchmarks, we deliberately specify latency constraints that are impossible to satisfy to also test the tool under extreme conditions. Setting α to 1 also does not result in any noticeable degradation

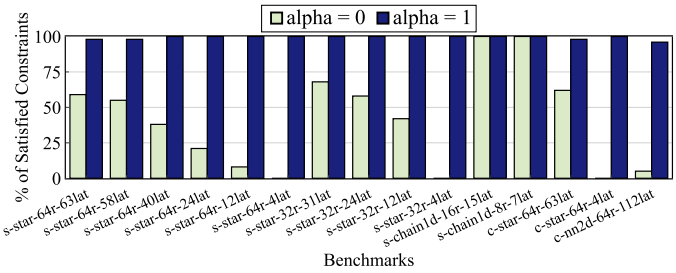

Fig. 7: Effect of the latency constraint cost term on the percentage of satisfied latency constraints.

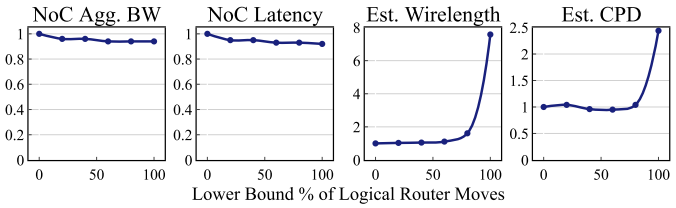

Fig. 8: Effect of the (lower bound) percentage of logical router moves during placement on different metrics. Setting this value in the range of 40-60% improves NoC aggregate bandwidth and latency without significantly impacting estimated wirelength and CPD.

in the post-placement estimated wirelength and CPD results. Following the same methodology, we select $\beta = 0.05$ which achieves a good trade-off between NoC performance metrics, wirelength, and CPD. We omit the detailed results for brevity.

2) Percentage of NoC Router Moves: We also study the effect of the number of logical router moves during placement on the quality of results. Since the number of logical router blocks can be a very small portion of all netlist blocks, randomly selecting a block for a placement move can result in fewer logical router moves than needed to fully optimize the NoC placement. Therefore, we counteract this by introducing a hard lower bound router move percentage ($P_{routers}$). For example, a lower bound percentage of 20% means that after every 4 block moves of any type selected by the placement reinforcement learning (RL) agent [35], a logical router block move is suggested. Fig. 8 shows the effect of increasing this lower bound percentage from 0% (baseline RL agent move selection) to 100% (only move logical router blocks) on different metrics. The results show that the NoC aggregate bandwidth and latency cost improve by 5-10% as more router moves are performed. On the other hand, the estimated wirelength and CPD are degraded as fewer placement moves are allocated to other netlist primitives, which negatively affects these metrics. A ratio of 20-60% of the placement moves dedicated to logical routers achieves a reasonable trade-off for these benchmarks. For the rest of this work, we manually set this value to 40% but in future work, we are planning to enhance the RL placement agent to also dynamically select the block type as well as the move type depending on the annealing stage. For example, the RL agent might find it more useful to suggest more router block moves earlier in the anneal, rather than later when moving a logical router location can be more disruptive to the placement solution.

3) NoC Placement Optimization Results: After empirically picking the cost function hyper-parameters ($\omega = 0.6$, $\alpha = 1$, $\beta = 0.05$) and the lower bound percentage of router moves ($P_{routers} = 0.4$), we run all our 27 complex synthetic benchmarks through the NoC-enhanced VPR flow and measure different placement quality metrics. Fig 9 shows the normalized results for the NoC aggregate bandwidth, NoC latency, post-placement estimated wirelength, and CPD. For the NoC aggregate bandwidth and latency, we also manually calculate the results for an optimal NoC placement of these benchmarks and compare them to our NoC placement optimization solutions. The results show that our NoC-enhanced VPR can optimize the NoC placement of these benchmarks and significantly reduce the NoC aggregate bandwidth and latency by a factor of 2× on average. As shown in the top two bar charts in Fig. 9, VPR results when optimizing NoC placement are very close to those of the optimal solution in most cases. The bottom two graphs of Fig. 9 also show that the added NoC placement optimization has little effect on the conventional placement quality metrics. The average increases in estimated wirelength and CPD are 5% and 1%, respectively.

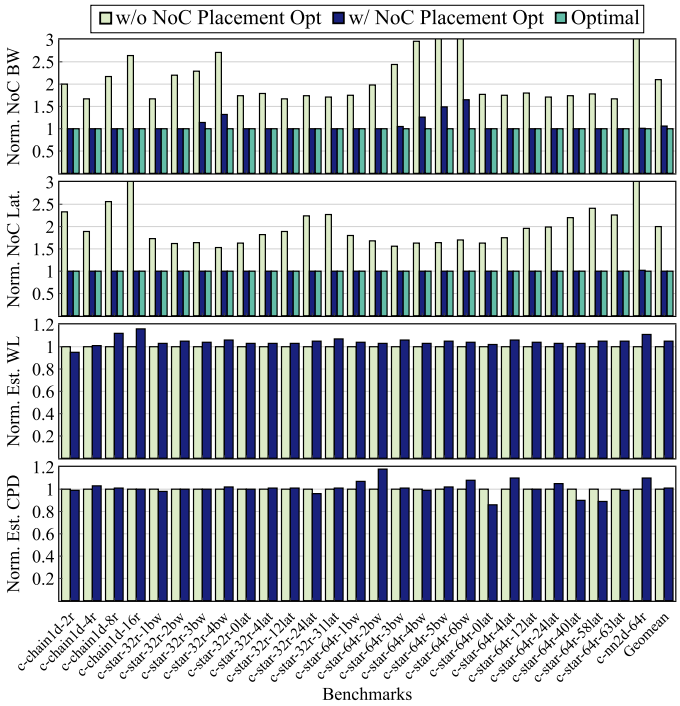

Fig. 9: Effect of NoC placement optimization on NoC aggregate bandwidth, NoC latency, estimated wirelength, and CPD for our complex synthetic benchmarks.

V. Case Study: RAD-Sim & NoC-Enhanced VPR

In this section, we present a case study that showcases our full flow shown in Fig. 2 with a real application design from the DL domain. We use RAD-Sim [9] to simulate the application design and automatically generate the traffic flows of real workloads to drive VPR's NoC placement optimization.

A. Application Design: Multi-Layer Perceptron Accelerator

For our case study, we develop a complete application design from the DL domain, both in SystemC to be simulated in RAD-Sim, and HDL to be pushed through the physical design flow. We implement a streaming-style accelerator for multi-layer perceptron (MLP) models that uses the NoC for communication between its compute engines. MLP models are feedforward neural networks in which each layer is a linear transformation of its input vector, followed by a non-linear

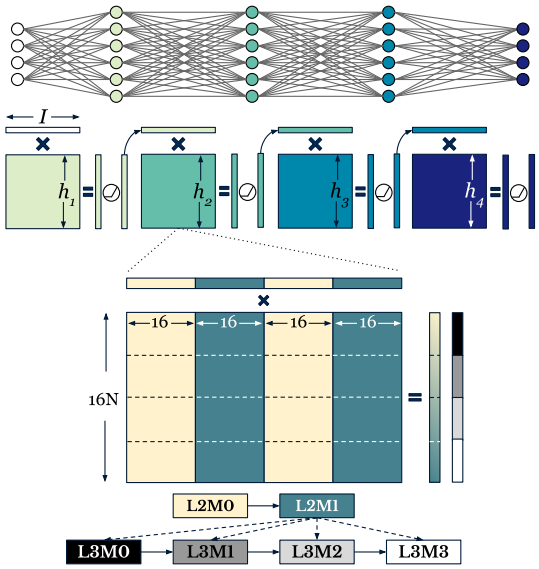

Fig. 10: Mapping of MLP to matrix-vector multiplications and mapping of each matrix-vector multiplication to MVM engines.

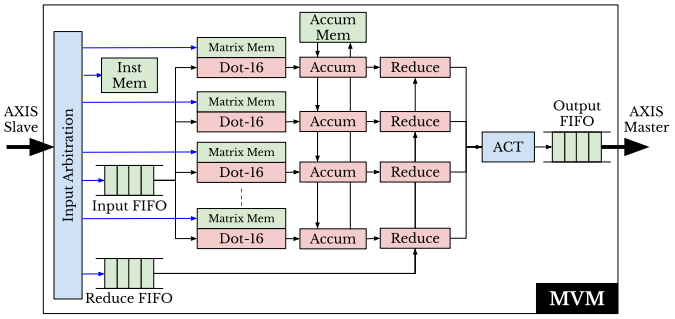

Fig. 11: MVM engine internal architecture.

TABLE I: MLP Acceleration test cases. I is the input vector size, H is the hidden dimensions of MLP layers, and LR is the total number of logical NoC routers in the design.

	Layers	*I*	*H*	MVMs/Layer	*LR*
MLP1	4	512	{512, 512, 256, 128}	{4, 3, 2, 2}	16
MLP2	2	512	{512, 512}	{4, 4}	13
MLP3	4	256	{256, 256, 256, 256}	{2, 2, 2, 2}	11
MLP4	3	1024	{512, 256, 128}	{2, 2, 2}	9

activation function (e.g. ReLU). Therefore, an MLP can be viewed as a sequence of back-to-back matrix-vector multiplications (MVMs) and non-linear activations, as illustrated in the upper part of Fig. 10.

To perform the computation of each layer, we tile the MVM into column blocks (highlighted with different colors in the bottom part of Fig. 10), and map alternating tiles to M_l different MVM compute engines. By doing that, all MVM engines dedicated to a specific layer can start processing different tiles in parallel, accumulate the partial results of different tiles over time steps, and finally reduce the accumulated partial results across MVM engines to produce the final result. The output is then passed through the activation function and distributed by the last MVM engine of layer l to the M_{l+1} MVM engines of the next layer $l+1$, as shown in Fig. 10 for $M_2 = 2$ and $M_3 = 4$. Our MVM engines are designed to fully parallelize the multiplication of a 16-element vector by a 16×16 matrix. If the matrix has more rows (e.g. $16N$ in Fig. 10), groups of 16 rows are processed sequentially over multiple time steps before processing the next column block.

Fig. 11 shows the internal architecture of an MVM engine. It has AXIS input/output interfaces to interface with the logical router IP discussed in Section IV. An input arbitration block steers incoming data to the instruction memory, input FIFO, or reduce FIFO depending on an associated tag value. The instruction memory stores program instruction words that orchestrate the execution of the MVM engine. The input and reduce FIFOs buffer the multiplication input vectors and partial result vectors for reduction across engines of the same layer, respectively. The MVM engine has 16 dot product engines, each of which is 16 lanes wide, to multiply a broadcast input vector by 16 distinct matrix rows from the local matrix memory banks. These are followed by accumulators to accumulate partial results from different matrix column blocks over time and store them in the accumulation memory. Then, the reduction units add the final accumulated results to those produced by another MVM in the same layer and buffered in the reduce FIFO. Finally, the results are concatenated, passed through the activation block, and buffered in the MVM output FIFO. When synthesized for a Stratix-IV-like architecture [33] with Arria-10-style DSP blocks supporting floating-point precision [39], a single MVM engine utilizes 17,441 logic elements, 288 DSPs, and 538 9Kb BRAMs. Besides the MVM engines, we implement input dispatcher modules that send initial input vectors to the MVMs of the first layer and a result collector module to which the last MVM of the last layer sends the final outputs.

We also implement a model compiler that takes as input an MLP description (number of layers, input size, hidden layer dimensions, number of MVM engines dedicated to each layer) and produces matrix memory initialization files, test inputs, and their corresponding golden results, and program instructions for each MVM engine such that all MVM engines collectively implement the described MLP. The outputs of this compiler are used to drive the RAD-Sim simulation and generate NoC traffic flows for realistic MLP workloads.

B. Experimental Results

Table I lists the parameters of the different MLP acceleration test cases that we experiment with. We first simulate all four test cases using RAD-Sim with a random NoC placement to generate the end-to-end application NoC traffic flows. Then, we pass the NoC traffic flow files and the HDL implementation of each test case through our physical design flow shown in Fig. 2 (our modified Titan flow for synthesis using Quartus followed by our NoC-enhanced VPR). Fig. 12 shows the effect of the NoC and circuit placement co-optimization (in comparison to no NoC placement optimization) on different implementation metrics. Fig. 12a and 12b show that our NoC-enhanced VPR can effectively optimize the NoC placement of large realistic designs and reduce the NoC aggregate bandwidth and latency by 2× and 1.6× on average, respectively. These improvements are achieved while preserving the circuit implementation quality. As shown in Fig. 12c and 12d, the final operating frequency and routed wirelength are within 3% of their values when no NoC placement optimization is performed. Finally, adding NoC placement optimization

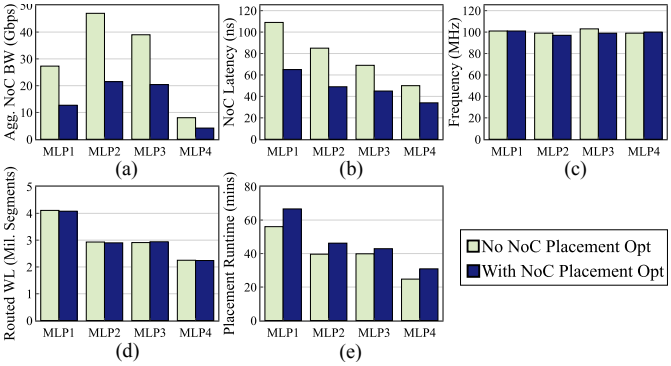

Fig. 12: Results for the MLP acceleration test cases with and without NoC placement optimization: (a) aggregate NoC bandwidth, (b) NoC latency, (c) post-routing frequency, (d) router wirelength, and (e) placement runtime. Our NoC-enhanced VPR optimizes NoC aggregate bandwidth and latency without affecting the final operating frequency and routed wirelength.

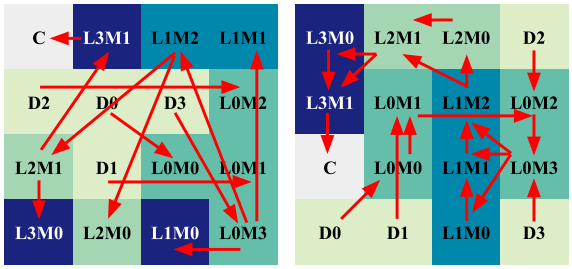

Fig. 13: Comparison of NoC traffic patterns (red arrows) of MLP1 test case when using a random NoC placement (left) vs. optimized NoC placement produced by VPR (right). D is input dispatcher, $L \bullet M \diamond$ is MVM $\diamond$ of layer $\bullet$, and C is result collector.

increases the placement runtime by 8-25% (17% on average) as shown in Fig. 12e. As expected, we can see that the optimized NoC placement produced by VPR results in much simpler and more regular NoC communication as shown by the visualizations generated by RAD-Sim in Fig. 13 for the MLP1 test case as an example.

C. The Benefit of Co-Optimization

The approach we present in this work integrates the NoC placement optimization into the conventional placement engine of the CAD flow. Another alternative is to perform a two-phase optimization in which the NoC placement is first optimized based on traffic flow information and then after NoC router locations are fixed, the rest of the circuit is placed using the conventional CAD tool placement engine. To the best of our knowledge, the two-phase approach is similar to that used by the NoC compiler in the AMD Vitis flow to optimize NoC placement for Versal devices [6], [29], although Vitis can iterate this 2-step flow multiple times in the case of legality or quality problems [30]. In this subsection, we evaluate the quality of results of both approaches using a variation of the MLP1 test case that uses a mix of programmable routing (between MVMs of the same layer) and NoC-based (between layers) communication.

To perform a two-phase optimization, we first run our NoC-enhanced VPR to optimize logical router locations with no logic attached to them using the application's traffic flows. Then, we fix these logical router locations to optimize NoC performance and run the tool again with the full MVM logic attached to the routers. Fig. 14 shows the results of this approach compared to our NoC and circuit placement co-optimization approach. Both approaches achieved the same NoC performance results, showing their effectiveness in finding performant NoC placement solutions. However, the final operating frequency and routed wirelength achieved by the two-phase optimization are 13% and 2% worse than that of the co-optimization approach, respectively. This highlights that some NoC placement solutions can result in a more difficult place and route problem for the rest of the circuit netlist, degrading the quality of results. Although the co-optimization approach increases placement time by 7% in this case, it resulted in a much easier routing problem that took approximately half the time to solve compared to the two-phase approach, as shown in the last two graphs of Fig. 14.

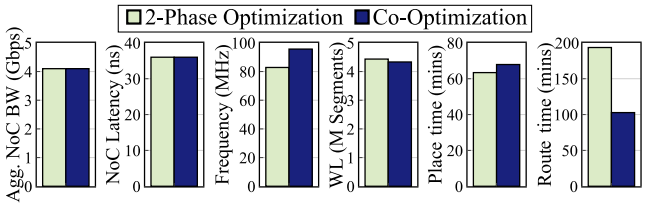

Fig. 14: Results for 2-phase placement optimization (NoC placement then circuit placement) vs. NoC & circuit placement co-optimization.

VI. Conclusion

In this paper, we enhanced the open-source VPR CAD flow to allow the exploration of new FPGA architectures with hard NoCs as well as new CAD algorithms for them. First, we extended the VPR FPGA architecture modeling language to describe embedded NoCs with different specifications and defined a traffic flow input file to capture a design's NoC communication patterns. Then, we modified the VPR placement engine to co-optimize NoC performance along with the conventional circuit metrics of wirelength and CPD. On a suite of synthetic benchmarks, the resulting flow achieves optimal or near-optimal NoC bandwidth and latency metrics, with little degradation in circuit wirelength or CPD.

Finally, we showcased the full flow integrating our NoC-enhanced VPR and RAD-Sim, a SystemC architectural simulator for novel reconfigurable acceleration devices. We use RAD-Sim to simulate four variations of an application design from the DL domain and generate NoC traffic flows for realistic workloads, which are then used to drive the NoC placement optimization in VPR. Enabling NoC placement optimization in VPR reduced the NoC aggregate bandwidth and latency of these application designs by 2× on average, with minimal impact on the achieved operating frequency and routed wirelength. We also show that our NoC and circuit placement co-optimization approach results in better quality of results and faster routing runtime compared to a two-phase approach in which the NoC placement is separately optimized first and then the remaining netlist placement is optimized after using a conventional FPGA CAD flow.

Acknowledgments

The authors thank the Intel/VMware Crossroads 3D-FPGA Research Center for funding support.

REFERENCES

[1] A. Boutros and V. Betz, "FPGA architecture: Principles and progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021.

[2] M. T. Bohr, "Interconnect Scaling - The Real Limiter to High Performance ULSI," in *Proceedings of International Electron Devices Meeting*. IEEE, 1995, pp. 241–244.

[3] M. Abbas and V. Betz, "Latency Insensitive Design Styles for FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 360–3607.

[4] S. Velagapudi and M. Honman, "Addressing Memory-Bandwidth and Compute-Intensive Challenges with Intel Agilex M-Series FPGAs (WP-01313-1.0)," 2022.

[5] M. S. Abdelfattah and V. Betz, "The Case for Embedded Networks on Chip on Field-Programmable Gate Arrays," *IEEE Micro*, vol. 34, no. 1, pp. 80–89, 2013.

[6] I. Swarbrick *et al.*, "Network-on-Chip Programmable Platform in Versal ACAP Architecture," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 212–221.

[7] Achronix Corp., "Speedster7t Network on Chip User Guide (UG089)," 2019.

[8] K. E. Murray *et al.*, "VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 1–55, 2020.

[9] A. Boutros *et al.*, "Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices," *IEEE Access*, vol. 10, pp. 95 067–95 082, 2022.

[10] M. Wissolik *et al.*, "Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance," *Xilinx Whitepaper*, 2017.

[11] Y.-k. Choi *et al.*, "HBM Connect: High-Performance HLS Interconnect for FPGA HBM," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 116–126.

[12] M. Langhammer *et al.*, "Spiderweb: High Performance FPGA NoC," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 115–118.

[13] N. Kapre and J. Gray, "Hoplite: Building Austere Overlay NoCs for FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[14] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs," in *ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA)*, 2012, pp. 37–46.

[15] M. S. Abdelfattah and V. Betz, "Networks-on-Chip for FPGAs: Hard, Soft or Mixed?" *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 3, pp. 1–22, 2014.

[16] ——, "Design Tradeoffs for Hard and Soft FPGA-based Networks-on-Chip," in *International Conference on Field-Programmable Technology (FPT)*, 2012, pp. 95–103.

[17] S. Yazdanshenas and V. Betz, "Interconnect Solutions for Virtualized Field-Programmable Gate Arrays," *IEEE Access*, vol. 6, pp. 10 497–10 507, 2018.

[18] B. Gaide *et al.*, "Xilinx Adaptive Compute Acceleration Platform: Versal Architecture," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 84–93.

[19] N. Jiang and other, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[20] M. Khairy *et al.*, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2020.

[21] G. Krishnan *et al.*, "SIAM: Chiplet-based Scalable In-Memory Acceleration with Mesh for Deep Neural Networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[22] M. S. Abdelfattah *et al.*, "Design and Applications for Embedded Networks-on-Chip on FPGAs," *IEEE Transactions on Computers*, vol. 66, no. 6, pp. 1008–1021, 2016.

[23] A. Boutros *et al.*, "RAD-Sim: Rapid Architecture Exploration for Novel Reconfigurable Acceleration Devices," in *arXiv:2301.04767*, 2023.

[24] C. Marcon *et al.*, "Exploring NoC Mapping Strategies: An Energy and Timing Aware Technique," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2005, pp. 502–507.

[25] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2004, pp. 896–901.

[26] J. Hu and R. Marculescu, "Energy- and Performance-Aware Mapping for Regular NoC Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 24, no. 4, pp. 551–562, 2005.

[27] S. Jagadheesh and P. V. Bhanu, "NoC Application Mapping Optimization using Reinforcement Learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2022.

[28] M. S. Abdelfattah and V. Betz, "LYNX: CAD for FPGA-Based Networks-on-Chip," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–10.

[29] I. Swarbrick *et al.*, "Versal Network-on-Chip (NoC)," in *IEEE Symposium on High-Performance Interconnects (HOTI)*, 2019, pp. 13–17.

[30] Advanced Micro Devices, Inc., "Versal ACAP Hardware, IP, and Platform Development Methodology Guide (UG1387)," 2022.

[31] P. Jamieson *et al.*, "Odin II: An Open-Source Verilog HDL Synthesis Tool for CAD Research," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 149–156.

[32] S. A. Damghani and K. B. Kent, "Yosys + Odin-II: The Odin-II Partial Mapper with Yosys Coarse-grained Netlists in VTR," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2022, pp. 157–157.

[33] K. E. Murray *et al.*, "Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap Between Academic and Commercial CAD," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, pp. 1–18, 2015.

[34] Mentor Graphics Corp., "Catapult High-Level Synthesis Datasheet (TECH15250-w)," 2017.

[35] M. A. Elgammal *et al.*, "RLPlace: Using Reinforcement Learning and Smart Perturbations to Optimize FPGA Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

[36] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 278–287, 1992.

[37] C.-C. Chang *et al.*, "Optimality and Scalability Study of Existing Placement Algorithms," in *Asia and South Pacific Design Automation Conference (ASP-DAC*, 2003.

[38] Verilog to Routing (VTR) Github Repository. [Online]. Available: https://github.com/verilog-to-routing/vtr-verilog-to-routing

[39] M. Langhammer and B. Pasca, "Floating-Point DSP Block Architecture for FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2015.