Hoplite-Q: Priority-Aware Routing in FPGA Overlay NoCs

Siddhartha
Nanyang Technological University
siddhart005@e.ntu.edu.sg

Nachiket Kapre
University of Waterloo
nachiket@uwaterloo.ca

***Abstract*—**
The Hoplite FPGA overlay network-on-chip routes packets in an oblivious manner without considering application priority when computing packet paths. This degrades performance across all priority classes of traffic by allowing them to interact and mix in the network in an arbitrary manner. However, real-world FPGA systems often need to route traffic from mixed-priority, multi-application workloads such as multi-tenant cloud deployments. Such scenarios require NoC resources to be allocated in a priority-aware manner to deliver expected Quality-of-Service outcomes to the FPGA applications. In this paper, we introduce Hoplite-Q, a lightweight router that exploits choice during routing to deliver improved outcomes for higher priority traffic on the NoC. We achieve this by (1) adding priority bits to the packet being routed, (2) enhancing routing choice in the switch with the addition of a single buffer, and (3) augmenting the routing function to use the buffer and priority tags in static and dynamic manner. Overall, the use of buffers and priority-aware routing improves throughput of high-priority applications by up to 1.8×, worst-case latency by 1.5–3.9×, while increasing the FPGA area utilization for the NoC by 1.3–3.8× on the Altera Arria 10 AX115S FPGA board.

I. Introduction

FPGA-based Network-on-Chip (NoC) solutions can provide high-bandwidth and low-latency connectivity between hardware components such as IP blocks and system-level interfaces such as PCIe endpoints, DRAM controllers, or Ethernet blocks on the FPGA in a resource-shared and cost-effective manner. NoCs can also be baked into accelerators as a communication fabric for routing traffic between spatial FPGA datapaths. In the datacenter domain, we foresee large NoC-enabled FPGA chips hosting multiple applications from different tenants with varying communication constraints on the same FPGA chip. In such scenarios, delivering different levels of service based on a cost-efficient policy could be desirable [16]. For example, NoCs being utilized in a shared environment could prioritize traffic based on a service tier and/or request priority of the customer.

Lightweight FPGA overlay NoCs such as Hoplite [8] provide an FPGA-optimized solution for composing large chip-wide systems scaling to thousands of processing elements and system interface ports. Hoplite is known to outperform competing FPGA NoCs such as CMU Connect [13] and Penn Split-Merge [6] designs by 1.5× in throughput (packets/cycle) while being 20–25× smaller in size, and 3–5× faster clock frequency. Hoplite is able to deliver this outcome through the use of deflection routing and a lean bufferless design. However, this low cost comes at a price: high deflection routing penalty for a single application, and an inability to discriminate between traffic from multiple distinct applications. In this paper, we address these issues by introducing Hoplite-Q, a priority-aware NoC router adapted from the original Hoplite router with support for different Quality-of-Service (QoS) outcomes for different traffic classes. To enhance routing choice in the Hoplite router, we add a *single* packet buffer inside the existing deflection router design. The buffer and associated multiplexing circuitry is crucial to unlocking the complete features of priority-aware routing in Hoplite. Our Hoplite-Q design then exploits this choice by using a suitably-designed priority driven routing function. The engineering considerations include choosing how to design the routing function to deliver the discrimination required between the different traffic classes.

The contributions in this paper are as follows:

- New design refinements of the Hoplite NoC router with a single buffer (Hoplite-B), and a fully-featured priority-aware design (static: Hoplite-Q, dynamic: Hoplite-Q*).
- Evaluation of router designs on various statistical and real-world benchmarks to quantify improvements in throughput, latency and cost.
- Evaluation of mixed-priority, multi-application workloads and measurement of associated QoS outcomes for statistical and real-world benchmarks.

II. Background

A. Hoplite NoC

The Hoplite [8] deflection router is a low-cost, FPGA-friendly NoC router that uses deflection routing coupled with an unidirectional 2D torus topology. It routes packets using a dimension-ordered routing (DOR) strategy, where packets are routed in the west-to-east direction (X-plane) first, before being routed along the north-to-south direction (Y-plane). The deflection router is also a bufferless switch design – packets are deflected, instead of buffered, whenever there is contention for a routing path inside a switch in any given cycle. This has a few implications: while the design is lightweight in LUT cost, packets can suffer from high

communication latencies due to deflections. Despite this, deflection routers can be competitive for latency-tolerant, throughput-sensitive real workloads, as their resource-light design is desirable for scalability.

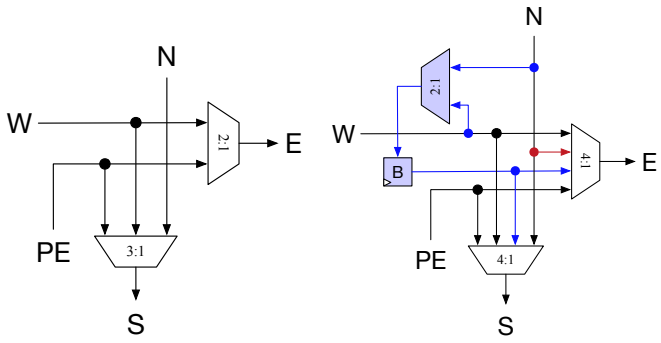
In Figure 1a, we show a high-level block diagram of the Hoplite router. Packets can enter the router from three input ports – the processing element (PE), west (W), and north (N) – and can exit from two output ports – east (E) and south (S). The S port is shared to deliver packets to the PE and a separate valid signal distinguishes this scenario. In the event of contention for an output port, the packet at input PE is given the lowest priority, *i.e.* no packet is accepted into the network from the PE in that cycle. If there is contention for the S output port, from valid packets at N and W input ports, then the N packet is always given routing priority, while the W packet is deflected to the E output port. This arbitration strategy trades off low packet latency for a lightweight bufferless switch design. The Hoplite deflection router is the baseline router design we build upon.

B. Related Work

Buffered routers ([1], [9], [2]) have long supported packet priorities to deliver QoS guarantees but are expensive to implement on top of FPGA fabric. In contrast, a majority of existing bufferless routers [3], [8], [10], [12] do not pay sufficient attention to delivering varying QoS for mixed-priority multi-application workloads. In fact, any priority-aware arbitration rules, such as the *Golden packet rule* [3], *Silver packet rule* [4] or *Oldest-First priority scheme* [10], are designed to reduce the number of total deflections and provide guarantees against livelock for the entire workload rather than individual application outcomes. MinBD [4] uses a side-buffer to reduce resource utilization and power costs associated with buffering, which is similar to our buffering strategies described later. Unlike the 4-flit side-buffer used in MinBD, we use a 1-flit (1-packet) deep buffer to reduce cost. HopliteRT [15] delivers QoS latency bounds with minimal hardware modifications, but is also incapable of distinguishing QoS demands across different priority bins.

III. Priority-Aware Hoplite

Modern NoCs support a range of routing algorithms which typically use destination address information to allow deadlock-free traversals on the network. We can add priority awareness to a NoC by simply adding additional priority bits, and using those bits along with the destination address bits to determine packet route. When used in buffered NoCs like in [1], [11], the lower priority packets can simply wait in the buffer a little longer. If we apply this naïvely to the Hoplite NoC, we observe that the routing algorithm has limited flexibility in making routing decisions due to the low arity of the undirectional torus switches. Thus, we must consider other adaptations to make priority-awareness more amenable to the Hoplite NoC design.



(a) Hoplite Router (b) Hoplite-Q Router

Figure 1: Hoplite-Q switch organization with enhancements. (1) Addition of priority bits in each packet, (2) Addition of a buffer B to store deflected W and N packets, (3) Enhanced choice during routing for selecting between W, N, PE, and B input ports.

In this section, we look at three main design considerations to support priority awareness in Hoplite NoC: (1) the design of a priority-aware routing function, (2) priority assignment method that may be static at compile time, or dynamic at runtime, and (3) introduction and sizing of the buffer in the switch to boost routing choice. We perform a careful cost-benefit analysis and quantify the effect of each proposed change.

A. Priority-Aware Routing Function

The dimension-ordered routing (DOR) function in Hoplite statically prioritizes N packets over W packets, while PE packets have the least priority. To support priority-awareness, at the very least, we must allow W packets to be able to deflect N packets. This necessitates the addition of a N→E turn in the switching crossbar (shown as the red turn in Figure 1b). For this configuration, we can design a priority-aware routing function on two NoC input ports – N and W. The less critical packet is always deflected to the E output port. Under this model, packets at PE are still given the least importance, regardless of their priority, *i.e.* just like Hoplite.

A fully-featured priority-aware router should be able to prioritize among all incoming packets regardless of their port of origin. Packet priority should be the sole merit in deciding packet routes. To alleviate the possibility of continually blocking the PE packet, we add a buffer in the router to hold N or W contending packets (shown in blue in Figure 1b), and allow the PE packets an opportunity to enter the NoC instead. We then adjust the routing function to consider four inputs at N, W, PE and B ports. With deflection-routed NoCs, the addition of such a buffer must be done with care as high buffering costs were the key

motivating factor behind the bufferless design of Hoplite. Our experiments show that a *single* buffer location delivers most of the benefits of a deeply buffered design, while increasing design cost modestly. We show a high-level view of the adaptations required to the original Hoplite NoC in Figure 1b (shown as blue multiplexers and registers).

Next, we must address two engineering decisions: (1) how do we update the packet priority bits, and (2) how do we determine which packets get to use the buffer. We answer these questions in the next three subsections.

B. Static Priority

For FPGA applications where communication pattern is known at compile time, we can identify priority classes at compile time. These are static priority bits that do not change value as they traverse the NoC.

- For applications with dataflow-type dependencies, the compute structure (dependency graph) can be extracted at compile time. We can run traditional slack analysis algorithms on these applications to determine the priority of each edge (or packet) in the graph. Such analysis reveals the critical paths in the application that must be prioritized for faster completion.
- For applications with no dataflow dependencies, such as Bulk-Synchronous Parallel workloads, there are no dependency chains or critical paths in the design. Instead, a Manhattan-distance heuristic can be used to estimate priority. A packet that must traverse a longer source-to-sink route is likely to suffer from more deflections at runtime.
- In multi-application scenarios, we assign priority to packets based on the quality of service desired by each application. In the extreme case, each application is assigned its own unique priority class. For example, if a 6-bit priority tag is used for two priority classes, packets in the lower and higher classes are assigned a priority value within $\{0,31\}$ and $\{32,63\}$ respectively. We focus on these mixed-priority multi-application scenarios for evaluation of the priority-aware Hoplite-Q router design.

Resource Impact: The resource impact of this design is the addition of P extra bits of priority wiring throughout the NoC. The ideal value of P will depend on system size, number of critical paths in the application, and number of applications using the NoC.

While this is a start, static priority alone is insufficient. Packets may get trapped in the buffer B and never have higher priority than any other packet seen thereafter. Static priority assignment may also lead to livelocks where packets with lower priority are deflected in perpetuity. Apart from livelocks, static priority does not account for NoC congestion effects and other dynamic events in the system. This motivates us to consider a dynamic adaptation to the priority function discussed next.

C. Dynamic Priority

For dynamic priority support, we explore the following two implementation strategies:

- We can increment packet priority on each *deflection* to account for the victimization suffered by a packet in the NoC. Thus, low-priority packets at injection can eventually attain high enough priority to exit the network. These low priority packets occupy valuable network capacity and should also be removed from the network to free up resources.
- We can also increment packet priority on each *hop* to account for the true age of a packet in the NoC. From our preliminary experiments, however, we found that this strategy is too aggressive and requires too many priority bits in the packet.

Overall, deflection-based priority updates result in better utilization of NoC resources and prevent denial-of-service (DoS) to lower priority packets. Dynamic priority updates are also applied to the packets waiting in the buffers so that packets do not get trapped in the buffers forever. When dynamic priority update support is enabled in Hoplite-Q, we denote that as Hoplite-Q* in this paper.

Resource Impact: The resource impact of this design is the introduction of P-bit adders in each router for all packets (outgoing links and buffer), and a potential increase in P to account for NoC traffic behavior.

D. Buffering

As discussed earlier, we need to add a buffer to Hoplite to increase the number of routing choices in the switch. This increased choice comes at the cost of ALMs and FFs to implement the buffer, and a 2:1 multiplexer for writing to the buffer. We empirically observe a depth of 1 to offer the best balance between cost increase and performance improvements. Packets will enter the buffer in the event of NoC conflict where two packets desire the same output port. Packets will leave the buffer when their priority is higher than other packets at the router. We also support *buffer redirection*, where a low-priority packet in a buffer is forcibly ejected to allow a higher-priority packet to be buffered instead of suffering a deflection. Redundant routes, such as PE$\rightarrow$B are also filtered out to ensure that the buffer is utilized efficiently.

With dynamic priority, we can ensure that waiting packets will eventually acquire sufficiently high priority to make progress in the NoC. Even in the absence of priority, the presence of a buffer can mitigate the effect of deflection penalties. If the injection rates in the NoC are moderate, the occasional buffering event can avoid the long deflection round-trips in the ring and boost performance. To separate the effect of buffering from priority, we consider a *single* buffer router (Hoplite-B) design, and compare its performance and area tradeoffs against the priority-aware Hoplite-Q(*) designs. In Hoplite-B, whenever there is contention for

the S port by packets at N and W, the packet at W is buffered. Since there is no priority-aware routing support in Hoplite-B, the priority of injection ports is fixed to N > W > B > PE, and N→E turns are not supported. A buffered packet in Hoplite-B would exit as soon as the S port is available.

Finally, the extra resources used by Hoplite-Q could also be theoretically used to create deeper buffers or replicated Hoplite channels. We test two new implementations to quantify these tradeoffs: (1) Hoplite-2B, which has a 2-deep buffer, and (2) Hoplite-2×, which is simply two identical and independent Hoplite channels.

IV. Methodology

A. RTL Implementation and Simulation

We compile all RTL code for all router designs with Altera Quartus Prime Standard Edition 16.0 targeting the Arria 10 AX115S device and generate post-fitting FPGA implementation metrics. We summarize the results in Table I. It is clear that the design adaptations to Hoplite cost extra resources. It is important to observe that the Hoplite-2× design which replicates the NoC doubles both the ALM and wiring cost. Hoplite-2B adds resources because of a larger buffer but preserves wiring cost. Hoplite-Q and Hoplite-Q* require a more complex arbitration function but are within 3–3.5× the ALM cost of baseline Hoplite while keeping wiring costs identical. As we will see later in Section V, the increase in resource cost gives us the priority-awareness properties we desire in our system.

We ran cycle-accurate simulations of the RTL using Verilator [14], which generates fast C++ code from the synthesizable RTL. Our C++ testbench can route traffic from various synthetic patterns, as well as communication traces from real-world workloads. For synthetic experiments, all PEs were configured to inject 2k single-flit packets at varying injection rates from 1% (1 packet injection attempt every 100 cycles on average) to 100% (1 packet injection attempt every cycle). We evaluated the performance of Hoplite-Q(*) on multi-application traces which simulate multiple instances of the application operating in different regions of the NoC. We explored various NoC system sizes: 1×1 (single PE) to 16×16 (256 PEs) configurations. Each packet carries a 32b payload and 8b address information along with P bits of priority information. Our experiments measured in-flight NoC latency, source queueing time, total packet latency, and sustained throughput metrics of the resulting implementation. We quantify the effect of system size, priority-tag bitwidth, static/dynamic priority-aware routing, along with variations due to real-world datasets.

Table I: Routers Resource Utilization (ALMs), 8b priority

Switch	Crossbar	%	Arbiter	%	Total
Hoplite	33	59	4	7	56
Hoplite-2×	72	60	10	8	121
Hoplite-B	34	43	8	10	80
Hoplite-2B	34	27	9	7	127
Hoplite-Q	40	22	108	61	178
Hoplite-Q*	40	19	127	59	215

Table II: Sparse matrix BSP benchmarks used in this paper

Benchmark	Domain	Nodes/Edges
`bp1600`	Simplex method basis matrix from the Harwell-Boeing Collection	822/4.8k
`mcca`	Non-LTE (local thermodynamic equilibrium) problem from astrophyics	180/2.7k
`simucaddac`	SPICE circuit simulation benchmark for 90nm process technology	654/5.5k
`lns511`	Fluid flow modeling benchmark	511/2.8k
`jpwh991`	Computer random simulation of a circuit physics model	991/6k
`add20`	Circuit netlist of a 20-bit adder	2.4k/17k

B. Benchmarks

We use sparse matrix-vector multiply (SpMV) benchmarks from a variety of domains and express them as Bulk Synchronous Parallel (BSP) [5] graphs. BSP is a well-known parallel compute abstraction that represents computation as a graph consisting of nodes (computation) connected by edges (communication). A BSP graph is evaluated in a synchronized lock-step fashion, where computation and communication occur in distinct stages separated by a global synchronization barrier. The NoC is utilized in the communication phase of the BSP workload, where a large number of packets are injected into the network and the system waits for all deliveries before proceeding. Such BSP applications are typically iterative applications that converge to a solution, and communication optimizations greatly influence program runtime. Table II summarizes the properties and the application domains of our benchmarks.

V. Results

In this section, we quantify the effect of priority-aware routing on synthetic and bulk-synchronous workloads.

A. Baseline Calibration Tests

First, we compare the performance and cost trends of baseline Hoplite against 1-deep (Hoplite-B) and 2-deep (Hoplite-2B) buffers, along with a 2-channel (Hoplite-2×) replicated solution.

In Figure 2, we quantify the resulting throughput and latency performance of Hoplite, Hoplite-B, Hoplite-2B and Hoplite-2× designs for `UNIFORM RANDOM` traffic at 64 PEs. Hoplite-B provides a significant improvement over Hoplite in throughput (1.5×) and latency performance (1.3×).

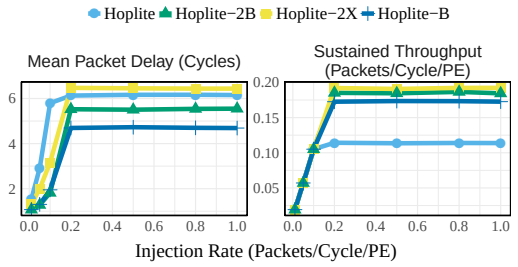


Figure 2: Mean packet latency and sustained throughput vs injection rate on 8×8 NoC (UNIFORM RANDOM)

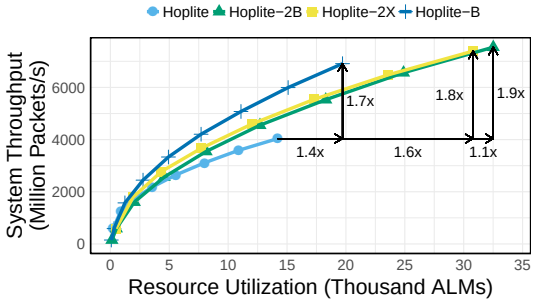


Figure 3: Sustained throughput (millions of packets per second) vs resource utilization (ALMs)

Hoplite-2B improves the throughput performance further but only marginally (1.6× over Hoplite, 1.1× over Hoplite-2B). Curiously, at an injection rate of ≥0.2, the average packet latency delay is higher in Hoplite-2B than Hoplite-B, despite higher sustained throughput. This effect can be attributed to the longer waiting time in deeper buffers. Overall, both Hoplite-B and Hoplite-2B improve NoC communication throughput performance by up to 60% by mitigating the deflection penalty suffered by packets. The dual-channel NoC, Hoplite-2×, achieves the highest throughput of all designs (1.7× over Hoplite), simply because packets can now be routed over two channels, instead of one, in parallel. The lack of buffers mean that packets still suffer long deflections in each channel, and hence, the average packet latency is close to that observed in single-channel Hoplite NoCs. The NoC saturation point is also higher for all three Hoplite-B, Hoplite-2B, and Hoplite-2× NoCs (0.2 packets/cycle/PE) when compared to baseline Hoplite NoC (0.15 packets/cycle/PE).

In Figure 3, we illustrate the tradeoffs between performance and logic resource utilization for these router designs after considering post fitting metrics (ALM cost, and Fmax). At 16×16 PEs, we observe a 70% improvement in throughput for an additional 40% resource cost when comparing Hoplite-B against baseline Hoplite NoC. Note that, for the target Arria 10 FPGA, the ALM resource utilization is still under 5% of the entire chip for the 16×16 NoC using Hoplite-B routers. The same comparison between Hoplite-2B and Hoplite-B highlights the diminishing performance returns – only an additional 10% improvement in throughput performance for 70% increase in resources used on the FPGA. This is supported previously by trends observed in [4], [7], where deeper buffers do provide improvements, but the most significant improvement comes from just one stage. Unlike [7] however, our solution is cheaper and does not use any BRAM resources and shares a single buffer location for holding deflected packets instead of a per-port buffer. Hoplite-2× offers a solution in between Hoplite-B and Hoplite-2B, but there is still an inefficient use of logic resources – 7% improvement in performance for 60% increase in resource utilization. Furthermore, increasing the number of communication channels on the NoC requires double the wiring resources that adds complexity to the overall design for limited benefit. Overall, the lightweight single buffer in Hoplite-B delivers the most effective balance between resource usage and increased performance.

B. *Effect of Buffering (Hoplite-B)*

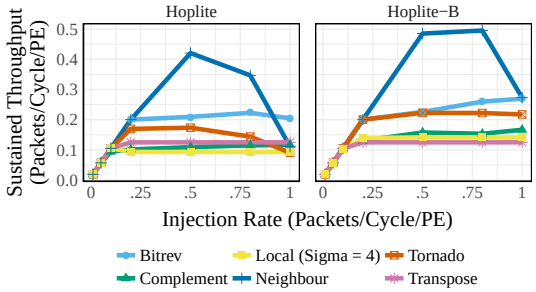
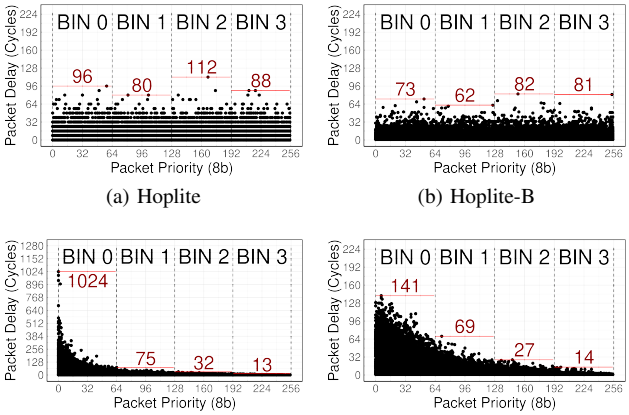


Figure 4: Sustained throughput vs injection rate for Hoplite and Hoplite-B under various traffic patterns on a 8×8 NoC.

In Figure 4, we compare the performance of a Hoplite NoC to a Hoplite-B NoC under various statistical traffic patterns for a system size of 8×8. The sustained throughput of all traffic patterns improves by 1.2–2.5×. We make the following observations:

The Hoplite-B router absorbs the deflections generated by the **Tornado** pattern to deliver ≈2.4× improvement (1.4× average) in overall sustained throughput. The **Neighbour** traffic pattern generates fewer deflections as traffic travels short distances. But even in this scenario, Hoplite-B, absorbs these deflections well and improves throughput by 1.3× across all injection rates. For the **Transpose** traffic pattern, both Hoplite and Hoplite-B router NoCs have similar throughputs. This is due to the presence of self-communicating PEs, *e.g.* (0,0) → (0,0) or (1,1) → (1,1) along the diagonal. However, the input PE port in the router is always given the least routing priority, which means that packets from these PEs contest for the shared exit/South port and are unable to exploit the buffer. For the **Bitrev**, **Local** and **Complement** traffic patterns, we observe up to 1.3–1.5× improvements in throughput at high injection rates.



(a) Hoplite (b) Hoplite-B

(c) Hoplite-Q (d) Hoplite-Q*

Figure 5: Packet delay distribution on 8×8 NoC for 4-application synthetic workload across various Hoplite NoC variants at 50% injection rate.

C. Effect of Priority (Hoplite-Q)

Next, we quantify the effect of priority-aware routing on a 4-application synthetic workload (`UNIFORM RANDOM` traffic, 32k packets) by measuring packet delay distributions in each priority class. Each packet is encoded with an 8b priority tag such that packets in priority class 0, 1, 2 and 3 are assigned a priority value within {0,63}, {64,127}, {128,191}, and {192,255} respectively. 8b adders are used to increment the priority value on deflection in Hoplite-Q*, and any overflow is prevented. In Figure 5, we see the results of this experiment.

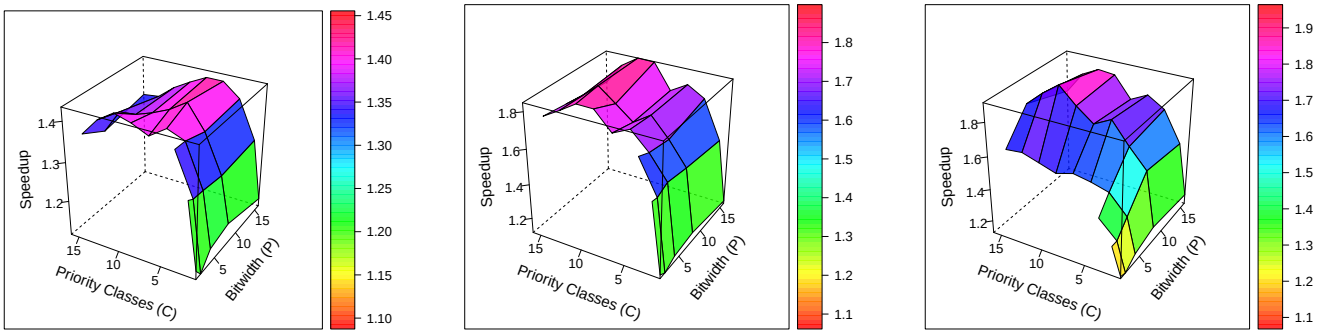
We track the number of *extra cycles* spent in the network by each packet due to deflections ($\text{Cycles}_{observed}$ - Cycles_{ideal}). Here, ideal cycles are computed simply as the Manhattan distance in absence of congestion. We then highlight the worst-case latency in each bin to quantify the observed QoS outcomes.

- In Figure 5a, the Hoplite router produces quantized levels of delay that are in multiples of 8. This is due to the 8×8 system size, and associated 8-cycle deflection in the X-plane. The worst-case latency delay suffered by a packet is 112 cycles, which indicates 14 deflections. We observe a uniform distribution of packet delay cycles in each priority bin as baseline Hoplite is oblivious to application priority.
- In contrast, Hoplite-B results in Figure 5b show a smoother packet delay distribution as packets may wait in the buffer upon deflection. The worst-case latency has improved to 82 cycles, but the packet delay distribution is still uniform across all priority bins.
- The Hoplite-Q NoC delays shown in Figure 5c indicate a priority-sensitive delay distribution across priority bins. The higher-priority bin now sees a worst-case delay of only 13 cycles while the least-priority bin sees a worst-case delay of up to 1024 cycles. Thus, better performance in the higher priority bin comes at the expense of lower performance in other bins. This packet delay distribution highlights the need for a dynamic priority-update feature that mitigates the observed aggressive starvation of low-priority packets.
- The Hoplite-Q* router design, shown in Figure 5d, offers a more balanced solution for this experiment by accounting for dynamic conditions in the NoC. The worst-case latency suffered by a packet in the highest-priority bin is now 14 cycles while that in the lowest-priority bin is a more reasonable 141 cycles. This marginally sacrifices the performance of the higher priority bins for balanced outcomes for other bins.

D. Priority Tag Bitwidth

One of the key design considerations is choosing the number of bits allocated to the priority field in the packet, especially with dynamic priority-updates in Hoplite-Q*. We now experiment with mixed-priority, multi-application BSP workloads where a varying number of applications are sharing the compute and communication resources available on the NoC. For fairness, we take a single real-world BSP application (`bp1600`) and replicate it multiple times while assigning a different priority class to each replicated copy. Figure 6 shows the throughput improvements observed on the *top-priority application* with Hoplite-B, Hoplite-Q, and Hoplite-Q* NoC when compared to baseline Hoplite.

As expected, since Hoplite-B does not route packets by priority, the performance of the top-priority application is similar to all applications in other priority classes. Increasing the number of bits (P) in the priority tag also has no noticeable effect on the performance. As the number of applications are increased, there is more congestion in the network and each router achieves its peak sustained throughput. Hoplite-B achieves a peak 1.4–1.5× speedup over Hoplite (see Figure 6a), which is expected and corroborated by results seen in Figure 2 earlier). Hoplite-Q is able to reserve and prioritize resources better for the top-priority application packets, and hence, deliver a higher peak of 1.7–1.8× throughput performance over Hoplite. Once again, since Hoplite-Q does not update packet priorities, there is no noticeable effect on increasing the bitwidth of the priority tag, *i.e.* we can remove the dynamic tag and choose P bits such that number of desired priority classes is $\leq 2^P$. Hoplite-Q* demonstrates a more noticeable change in performance with varying P. At small P, the numerical range of the dynamic tag is not large enough, such that deflected packets from the lower-priority classes get quickly promoted to higher priority classes and limit the performance advantage reserved for the top-priority application. At large P, the effect of the dynamic priority-updates is diminished, as it takes many more deflections for packets to get promoted across different priority classes. This results in more low-priority packets getting stuck for long periods in the NoC, which takes up routing resources (*e.g.* buffers)



(a) Hoplite-B vs Hoplite (b) Hoplite-Q vs Hoplite (c) Hoplite-Q* vs Hoplite

Figure 6: Observed throughput improvements over 8×8 Hoplite NoC for the application in the top-priority class, tested over varying number of priority classes (C = 1→16), and total bitwidth of the priority tag (P = 1→16).

and, on the whole, throttles the performance of the NoC. The design optimum seems to be at P = 8 for the range of priority classes tested in our experiments, and is likely to be larger as we scale further up beyond 16 priority classes. When compared to Hoplite-B and Hoplite-Q, Hoplite-Q* also delivers a slightly higher peak throughput performance of 1.8–1.9× over Hoplite at P = 8.

E. Throughput vs Average Latency Trends

In this subsection, we look more closely at the effect of priority-aware routing on packets from all priority classes. All results detailed in this subsection, unless otherwise stated, are evaluated with the `bp1600` BSP benchmark. All latency measurements visualized in this section include both injection queue latency and packet in-flight latency.

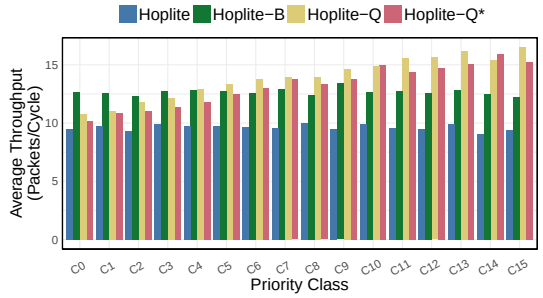


Figure 7: Average throughput of application in each priority class on 8×8 NoC, where C = 16 and P = 8.

Figure 7 shows the effect each router has on the throughput of applications in each priority class (labeled C0 to C15, in increasing order of priority). As expected, Hoplite and Hoplite-B NoCs produce a fairly uniform trend across all priority classes since they do not have any priority-aware routing features. Hoplite-Q produces a stark upward trend in the throughput performance from C0→C15. Hoplite-Q* distributes this effect slightly to produce a better balanced throughput performance across all priority bins. When comparing throughput performance between C0 and C15, there is a ≈60% improvement in throughput for the top-priority application. This throughput performance is supported by Figure 8 as well, where we track and compute the average packet latency suffered by packets in each priority class. Here, the quality of service provided to top-priority packets (C15) can be almost 4× better than lowest-priority packets (C0). Note that an overwhelmingly large proportion of the routing latency is due to the injection queue at the PE input, and hence, the eventual throughput improvements across priority bins saturate at smaller values.

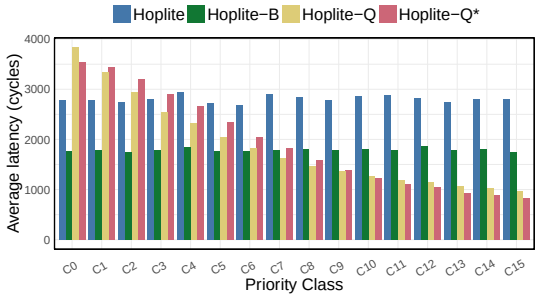


Figure 8: Average latency suffered by packets in each priority class for 8×8 NoC, where C = 16 and P = 8.

Figure 9 shows the throughput and packet latency performance of the *top-priority* application for different benchmarks. Each benchmark has its own communication pattern, and hence has a varying response to Hoplite-Q enhancements. Since larger benchmarks have more communication edges, Hoplite-Q(*) has better opportunities to improve average latency of packets in the top-priority workload.

Finally, in Figure 10, we observe that the gap in throughput performance of Hoplite and Hoplite-B routers increases as system size increases. This is because the deflection penalty at large system sizes impacts performance more severely. The scaling trend of Hoplite-Q(*) demonstrates that the router does not sacrifice any advantages of Hoplite-B, but instead utilizes all available resources efficiently to deliver

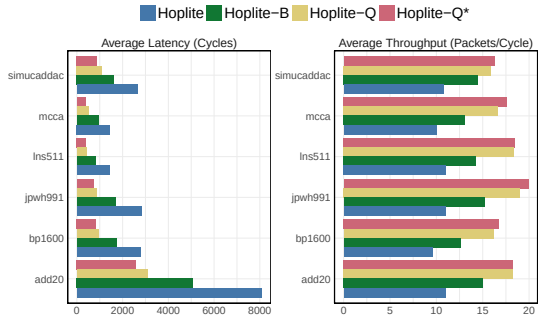


Figure 9: Average throughput and packet latency of application in top-priority class across different BSP benchmarks for an 8×8 NoC, where C = 16 and P = 8.

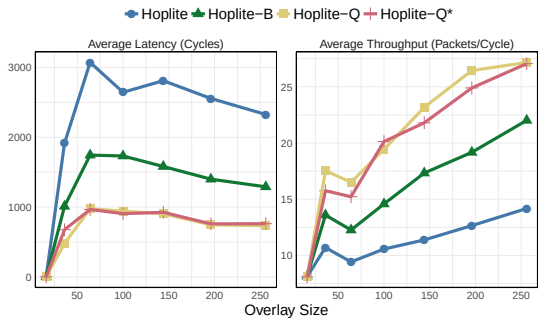


Figure 10: Average throughput and packet latency for top-priority application vs overlay size. C = 16, P = 8.

desirable QoS outcomes for the top-priority application.

VI. Conclusions

We introduce Hoplite-Q, a lightweight FPGA-friendly priority-aware NoC router capable of exploiting application priority when routing communication workloads. We modify the Hoplite FPGA NoC router by adding a single buffer to enhance routing choice and redesign the routing function to use the priority bits in each packet when determining packet paths. Our proposed design improves throughput of the top-priority workload in the mix by 1.3–1.8×, and worst-case latency by 1.5–3.9×, while increasing FPGA area cost by 3.8×. Hoplite-Q paves the way for a tiered model in multi-tenant datacenter FPGA deployments, where traffic from different tenants can be serviced differently based on the Quality-of-Service (QoS) desired.

References

[1] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Aergia: A Network-on-Chip Exploiting Packet Latency Slack. *Micro*, 31(1):29–41, Jan 2011.

[2] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Application-aware prioritization mechanisms for on-chip networks. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 280–291. IEEE, 2009.

[3] C. Fallin, C. Craik, and O. Mutlu. CHIPPER: A low-complexity bufferless deflection router. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symp.*, pages 144–155. IEEE, 2011.

[4] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu. MinBD: Minimally-buffered deflection routing for energy-efficient interconnect. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 1–10. IEEE, 2012.

[5] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, 22(2):251–267, 1994.

[6] Y. Huan and A. DeHon. FPGA optimized packet-switched NoC using split and merge primitives. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pages 47–52, Dec. 2012.

[7] N. Kapre. Implementing FPGA Overlay NoCs Using the Xilinx UltraScale Memory Cascades. In *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*, pages 40–47. IEEE, 2017.

[8] N. Kapre and J. Gray. Hoplite: Building austere overlay NoCs for FPGAs. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, Sept 2015.

[9] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 89–100. IEEE Computer Society, 2008.

[10] Y. Li, K. Mei, Y. Liu, N. Zheng, and Y. Xu. LDBR: Low-deflection bufferless router for cost-sensitive network-on-chip design. *Microprocessors and Microsystems*, 38(7):669–680, 2014.

[11] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of on-chip networks using deflection routing. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pages 296–301. ACM, 2006.

[12] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 196–207. ACM, 2009.

[13] M. K. Papamichael and J. C. Hoe. CONNECT: re-examining conventional wisdom for designing NoCs in the context of FPGAs. In *the ACM/SIGDA international symposium*, page 37, New York, New York, USA, 2012. ACM Press.

[14] W. Snyder. Verilator and systemperl. In *North American SystemC Users' Group, Design Automation Conference*, 2004.

[15] S. Wasly, R. Pellizzoni, and N. Kapre. Worst case latency analysis for hoplite FPGA-based NoC. Technical report, 2017.

[16] H. Yuan, J. Bi, W. Tan, and B. H. Li. CAWSAC: Cost-aware workload scheduling and admission control for distributed cloud data centers. *IEEE Transactions on Automation Science and Engineering*, 13(2):976–985, 2016.