Learn the Switches: Evolving FPGA NoCs with Stall-Free and Backpressure based Routers

Gurshaant Malik
gsmalik@uwaterloo.ca

Ian Elmor Lang
ielmorlang@uwaterloo.ca

Rodolfo Pellizoni
rpellizz@uwaterloo.ca

Nachiket Kapre
nachiket@uwaterloo.ca

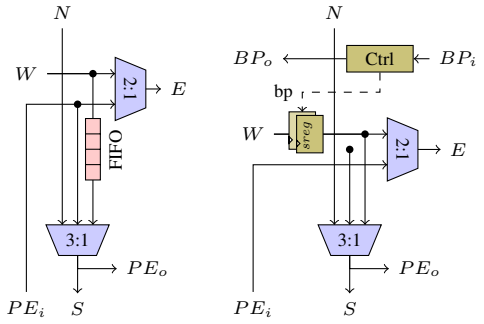
Abstract*—We can overcome the pessimism in worst-case routing latency analysis of timing-predictable Network-on-Chip (NoC) workloads by single digit factors through the use of a hybrid FPGA-optimized NoC. Timing-predictable FPGA-optimized NoCs such as HopliteBuf integrate stall-free FIFOs that are sized using offline, static analysis of a user-supplied flow pattern and rates. For certain bursty traffic and flow configurations, the static analysis delivers very large, sometimes infeasible, FIFO size bounds and large worst-case latency bounds. Alternatively, backpressure-based NoCs such as HopliteBP can operate with lower latencies for certain bursty flows. However, they suffer from severe pessimism in the analysis due to the effect of pipelining of packets and interleaving of flows at switch ports. As we show in this paper, a hybrid FPGA NoC that seamlessly composes both design styles on a per-switch basis, delivers the best of both worlds with improved feasibility (bounded operation), and tighter latency bounds. We select the NoC switch configuration though a novel evolutionary algorithm based on Maximum Likelihood Estimation (MLE). For synthetic (`RANDOM`, `LOCAL`) and real world (`SpMV`, `Graph`) workloads, we demonstrate ≈2–3× improvements in feasibility, ≈1–6.8× in worst-case latency while only requiring LUT cost ≈1–1.5× larger than the cheapest HopliteBuf solution. We also deploy and verify our NoC (PL) and MLE framework (PS) on a Pynq-Z1 to adapt and reconfigure NoC switches dynamically.

I. Introduction

With the growing communication demands of modern FPGA system-level interfaces like HBM stacks, high-speed networking, and multi-chip module IOs, it is imperative that we support data movement using resource-shared, high-performance NoCs. The Xilinx Versal NoC [18] is a *hard* network-on-chip that is permanently embedded in the FPGA fabric with fixed bandwidth and routing features that are tailored for distributing high-speed HBM and high-speed IO interface bandwidth across the FPGA fabric. Hoplite [9], CMU CONNECT [16], and Penn Split-Merge [7] switches are *soft* network-on-chip architectures that can be implemented using existing FPGA LUTs and interconnect. A combination of both styles of NoCs will be necessary to address data movement requirements that span the entire FPGA die including last-mile connectivity. Regardless of the style of NoC used, there is a need for mapping tools and analysis techniques for making efficient use of these communication structures. In this paper, we develop mapping tools targeting *soft* NoCs or *configurable, hard* NoC switches that allow customization of NoC operation on a per-switch basis.

The Hoplite [9] FPGA NoC is a LUT-optimized network-on-chip architecture targeting fracturable Xilinx FPGAs for high-speed, low-cost operation. Several variants of Hoplite targeting efficient implementations with different cost-feature tradeoffs such as HopliteRT [19], and HopliteBuf [4] have been published. These designs eliminate the livelock limitations of the original Hoplite design and provide provable upper bounds on packet latency. This is crucial for safety-critical real-time systems, where timing properties of the underlying hardware are used to ensure that applications meet their scheduling deadlines, and performance isolation between different communicating components is required [8]. Of particular interest is the HopliteBuf variant that adds stall-free SRL32 FIFOs to the switch and uses static analysis tools to prove upper bounds on FIFO sizes and worst-case routing latency (`wclatency`). However, under certain scenarios, the static analysis exaggerates FIFO and latency bounds making them impractical for real designs. Under these circumstances, a different variant of the NoC with lightweight backpressure, HopliteBP [5] may be preferred. If the entire NoC adopts a backpressure-based routing style, the blocking effects of backpressure due to pipelining and interleaving of flows will severely limit provable NoC performance [5]. Furthermore, the FPGA implementation requirements of flow control, however lightweight they may be, were a primary motivation for the deflection-oriented design of Hoplite. Instead of a homogeneous NoC design, a carefully selected hybrid NoC architecture that combines both HopliteBuf and HopliteBP styles in a fine-grained fashion yields a superior solution than either alternatives alone.

We show a set of three network flows on a 4×4 NoC interacting in prescribed ways in Fig. 2. HopliteBuf provides better worst-case latency for the horizontal traffic in Figure 2a, as HopliteBP suffers from backpressure propagated on the horizontal connections. However, for vertical traffic in Figure 2b, HopliteBP delivers 1.5× better worst-case latency. The cyclic loop of dependencies between flows shown in Figure 2c further affects HopliteBuf, reducing its range of statically analyzable rates, while HopliteBP is less affected. We analyze this example in further detail in Section II-B. Based on these observations, it is clear that one-style-fits-all approach will not work and we need to configure each switch in the NoC carefully by learning the effect of interactions between the conflicting traffic flows of the application. As the interference pattern of the network flows can be quite complex, and a brute-force approach not feasible for large NoC sizes, we develop a evolutionary strategy to discover high-quality solutions for



(a) HopliteBuf (FIFO) (b) HopliteBP (Backpressure)

Fig. 1: Block diagrams of HopliteBuf and HopliteBP designs.

a given QoR function. We formulate a Maximum Likelihood Estimation solution where we adjust the probabilities of each switch configuration based on iterative analysis trials. The key contributions of this paper include:

- Design of a hybrid FPGA NoC architecture that combines HopliteBuf and HopliteBP switches to exploit the best of both worlds.
- Development of an evolutionary strategy to learn switch configurations for a defined QoR function using Maximum Likelihood Estimation (MLE), scalable to large NoC sizes.
- Quantification of worst-case latency, feasibility, cost and optimization runtime across real and synthetic applications, demonstrating a ≈1–6.8× reduction in worst-case latency and ≈2-3× improved feasibility.
- We implement and verify our entire framework on a Pynq-Z1. We implement the MLE optimization tool on the Cortex A9 processor (PS) to configure the FPGA NoC switches dynamically, while realizing the NoC itself on the FPGA.

II. Switch Design

In this section, we discuss the design of the existing HopliteBuf and HopliteBP networks, motivating examples for a hybrid design, and details of FPGA implementation.

A. Background: HopliteBuf and HopliteBP

The FPGA NoC switches (see Fig. 1) explored in this paper are based on the Hoplite [9], [10] design. It is an FPGA-optimized switch that is integrated in a unidirectional torus topology and routes packets using deflections rather than buffering flow control. Packets use DOR (dimension-ordered routing) policy where they traverse in the X-dimension ($W \rightarrow E$) first before turning ($W \rightarrow S$) into the Y-dimension ($N \rightarrow S$). The possibility of routing livelock and out of order packet delivery are significant limitation of Hoplite, making it unsuitable for systems requiring guarantees on worst-case packet latencies.

HopliteBuf (FIFO in Fig. 1a) [4] is a variant of Hoplite that introduces stall-free FIFOs on the turns in the NoC. We buffer packets turning from $W \rightarrow S$ if a $N \rightarrow S$ packet is present in that cycle. Following DOR routing policy, we allow N packets to travel S while W packets must wait for an empty cycle to progress further. The 2:1 East mux chooses between W, and PE packets while the South mux chooses between W' (FIFO output), N, and PE packets. We assume that packet delivery to a PE is not stallable allowing unrestricted N→S traffic.

HopliteBP (Backpressure in Fig. 1b) [5] supports lightweight backpressure in the horizontal ring. $N \rightarrow S$ packets have the highest priority; thus foregoing a need for backpressure along the vertical dimension. Only packets turning $W \rightarrow S$ in the network may be subject to contention and thus require a backpressure interface. We insert shadow registers on the W port which provide an ability to store stalled packets *in place* and propagate the backpressure control upstream in the opposite direction of packet flow [15].

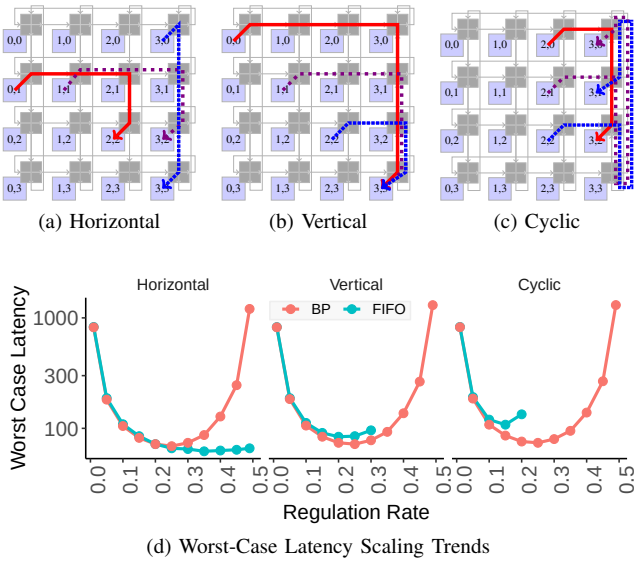
HopliteBuf+BP is a unified switch that includes both switch components and supports runtime configuration of mode. This mimics the behavior of a configurable hard NoC, and allows soft NoCs the ability to adapt dynamically to new application needs without requiring full FPGA bitstream reconfiguration.

Both HopliteBuf and HopliteBP switches are complemented with a static analysis and traffic regulation component [5]. We leverage this analysis to prove feasibility (bounded latencies and FIFO sizes), compute worst-case FIFO size (for HopliteBuf), as well as worst-case bounds on latency (injection+routing) of packets. The analysis computes *composable* latency bounds, that is, the bounds do not depend on detailed information about activity of unrelated PEs. To this end, it limits the maximum number of packets that a PE can inject in the network in any interval of time using a network *regulator*. In this way, to compute the latency for flow f_i, it only needs to know the regulation parameters and source/destination PEs for every other flow in its set of interfering flows. The analysis employs a leaky bucket regulator [13], which uses two parameters, regulation rate $\mathbf{f_i}.\boldsymbol{\rho}$ and data block size $\mathbf{f_i}.\mathbf{b}$: data block is the maximum number of consecutive packets that the flow can send through the regulator, while the rate is the maximum long-term throughput of the regulator in packets per cycle. For a more in-depth discussions of regulation and bound mathematics, please refer to [5].

B. Motivating Example

To discuss how the switch configuration affects the results of the analysis for different flow patterns, we perform a set of analyses for the three flow sets Horizontal (a), Vertical (b) and Cyclic (c) shown in Figure 2. We assume that all flows have the same block size $f_i.b = 8$ and are equally important, hence we use the same value of regulation rate $f_i.\rho$ across all flows.

In Figure 2d, we then plot the maximum analytical latency of the NoC while varying $f_i.\rho$. We consider two network configurations, one where all switches use stall-free FIFOs (HopliteBuf), and one where all switches use backpressure (HopliteBP). We stop plotting the latency whenever the network declares infeasibility due to unbounded latencies or the size of any buffer under HopliteBuf exceeds 32 (as this enables one-LUT per bit FIFO implementation).



(a) Horizontal (b) Vertical (c) Cyclic

(d) Worst-Case Latency Scaling Trends

Fig. 2: Interacting network flows on a 4×4 NoC arranged to show the benefits of mixing FIFO and Backpressure switches.

- **Effect of regulation rate**: As we increase regulation rate, latency initially decreases for all cases in Fig. 2d. This is because decreasing the regulation period ($\frac{1}{f_i.\rho}$) affords each PE increased opportunity to inject its packets into the network. However, past a certain regulation rate, the latency starts increasing due to increased contention caused by conflicting flows, which dominates any benefits of a reduced regulation period. Increasing the regulation rate past such point can still be useful, as it allows the flows to inject a higher number of packets in the average case. In this paper, we focus on optimizing the network configuration's QoR assuming that the importance, and thus regulation rate, of each flow is given by a system designer.
- **Horizontal**: For the set of flows shown in Fig. 2a, we note that HopliteBuf (fifo) delivers lower worst-case latency. This is because under HopliteBP, f_1 (▬) suffers interference not only from f_2(▪ ▪) but also f_3 (▪▪▪) indirectly due to horizontal backpressure stalls originating from switch $(3,1)$; thus increasing latency compared to HopliteBuf.
- **Vertical**: Here, HopliteBP achieves lower latency. This is due to the effects of buffering: since f_2 (▪ ▪) suffers interference from f_1 (▬), HopliteBuf will accumulate its packets at switch $(3,1)$. Hence, the maximum number of consecutive packets that f_2 (▪ ▪) can inject south at $(3,1)$ becomes larger than its data block size $f_2.b$. In turn, this means that the queueing delay of f_3 (▪▪▪) at switch $(3,2)$'s FIFO can exceed injection delay at the source of f_3 (▪▪▪) due to backpressure induced stalls in HopliteBP.
- **Cyclic**: The negative effects of vertical buffering are magnified in a cyclic flow pattern and the HopliteBuf designs show significantly worse latency than HopliteBP, as shown in Fig. 2d.

A user-supplied set of network flows can contain a combination of these three patterns, among others. The key idea explored in this paper is to learn the effect of these flowset interactions on routability and hardware costs to make a determination between HopliteBuf and HopliteBP on a *per-switch* basis. The use of such hybrid NoC switch configurations allows to optimize both analytical flow latency as well as hardware cost. This in-turn allows us to tailor our NoC offerings to the application at-hand and QoR requested, by learning the effects of the provided flows on a *per-switch* basis.

C. Xilinx FPGA Mapping

The muxes of the original Hoplite switch [19] can be efficiently mapped to a single fracturable 6-LUT for one bit of switching datapath. For the HopliteBuf design, we are unable to exploit that degree of compactness as the south mux needs to select between three inputs: N, FIFO output and PE_i. To realize these FIFOs, we can make use of the SRL32 primitive on Xilinx FPGAs that repurposes LUTs as Memory elements. For HopliteBP, the DOR routing logic must be adapted to account for the presence of backpressure signals. The control logic and shadow registers are more expensive to implement on the FPGA than SRL-based FIFOs. They result in a 1.2–1.8× increase in LUT and FF usage over HopliteBuf switches as shown in Table I. We also build a unified HopliteBuf+BP switch that not only permits propagation of backpressure when composing a mixture of HopliteBuf and HopliteBP switches, but also provides runtime adaptability to choose either operating mode.

	Hoplite	HopliteBuf	HopliteBP	HopliteBuf+BP
LUTs	59	161	189	247
FFs	86	91	167	175

TABLE I: Resource utilization of Hoplite based switches on Xilinx Virtex-7 for 32b datawidth and 32-deep SRL32 FIFOs.

III. Evolutionary Learning of NoC Switches

The key idea explored in this paper is the use of hybrid NoC switch configurations that mix HopliteBuf and HopliteBP styles in a single NoC. Depending on the interference pattern of network flows, each switch configuration may be tailored in an application-specific manner. In Figure 3, we show the best switch configuration for a simple 3×3 NoC with a fixed set of nine flows (shown in red in the leftmost NoC subfigure). We vary the block size b and regulation rate ρ for each flow (identically). At very low rates and data block sizes, HopliteBuf offers the cheapest and best-performing design, while at larger rates and blocks, the design starts to migrate to HopliteBP-dominated solutions. Given that an $N \times N$ NoC will have N^2 switches, each with a boolean decision to make, the design space grows exponentially with problem size.

Now, we discuss the optimization technique we use to learn the switch configurations for a particular set of flows. Specifically, given the regulation rate and data block size of each flow, we determine whether each switch in the network implements a stall-free FIFO or backpressure logic to optimise

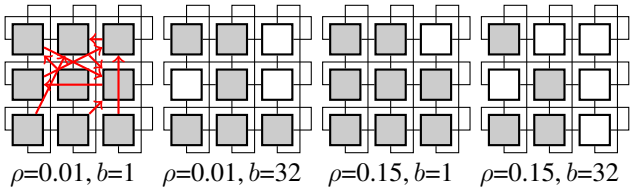


Fig. 3: FPGA NoC Switch Configuration for a 3×3 NoC for a set of nine flows (shown in leftmost subfigure) with identical but four combinations of rate ρ and block size b characteristics. At low rates and block sizes, most switches tend to be cheap HopliteBuf ◻ variants, while a growing number convert to expensive HopliteBP ◻ variants with increasing rate+size.

for the user defined QoR (`wclatency*cost`, `wclatency`, etc).

Note that an $N \times N$ NoC has N^2 switches, each with a boolean decision to make. This $O(2^{N^2})$ solution space necessitates the use of a scalable approach rather than brute-force exploration (a 7 × 7 NoC can have ≈562 trillion solutions). We choose to model this as a Maximum Likelihood Estimation algorithm. We model each switch of the NoC as an independent random variable from a *2-point Bernoulli distribution* B_i. This means that there are two outcomes possible for each switch: 0 represents a HopliteBuf and a 1 represents HopliteBP. For each switch i, we represent the probability of choosing a Backpressure configuration as p_i while the FIFO choice becomes $1 - p_i$. We seed each switch with zero bias by ensuring that the skew of each switch's distribution B_i, $\frac{1-2p_i}{\sqrt{(p_i(1-p_i)}}$, is 0 by starting with p_i = 0.5 ∀ i.

We aim to *evolve* the optimal NoC configuration by producing multiple candidates solutions for the g^{th} generational step for each switch s_i^g of the $N \times N$ NoC. Unlike conventional neural networks with a known training set, we generate our training set on-the-fly based on the results of the generational search. Unlike Naïve Bayesian inference strategies explored by InTime [11], our approach directly aims to minimize an objective function rather than train with a binary classifier. For our setup, we evolve by sampling each switch's Bernoulli distribution $B_i^g(p_i^g)$ to produce either a FIFO (0) or a Backpressure configured (1) switch. For each generation step g, we produce C potential candidate NoC configurations $H_C^g \in (0,1)^{N^2 \times C}$, as shown below in Equation 1. For our experiments we chose C=100.

$$H_C^g = \underbrace{\begin{bmatrix} s^g_{1,1} & s^g_{2,1} & \cdots & s^g_{C,1} \\ \vdots & \vdots & \ddots & \vdots \\ s^g_{1,N^2} & s^g_{2,N^2} & \cdots & s^g_{C,N^2} \end{bmatrix}}_{\text{Each } \textbf{column} \text{ is a flattened NoC configuration } (h_n)} \quad (1)$$

We test each candidate configuration h_n for fitness on a user defined function and filter out the top λ=25% performing candidates $H_{\lambda:C}^g \in (0,1)^{N^2 \times \lambda}$. This is a greedy step but the memory of previous iterations is reflected in the existing probabilities of the switch configurations. We then adapt each switch's Bernoulli distribution in the general direction of the chosen candidates. We aim to increase the likelihood that the top performing candidates $H_{\lambda:C}^g$ of this generation were sampled from it. For each switch s_i^g at generation g, we define a likelihood function as follows:

$$\begin{aligned} f\left(s^g_{1:C,i},\ldots s^g_{\lambda:C,i}|p_i^{g+1}\right) &= P\left(s^g_{1:C,i}, s^g_{2:C,i}\ldots, s^g_{\lambda:C,i}|p_i^{g+1}\right) \\ &= \underbrace{P(x_1, x_2\ldots, x_\lambda|\hat{p})}_{\text{substitute}} \\ &= \hat{p}^{x_1} \cdot (1-\hat{p})^{1-x_1} \ldots \hat{p}^{x_\lambda} \cdot (1-\hat{p})^{1-x_\lambda} \\ &= \hat{p}^{\sum^{\lambda} x} \cdot (1-\hat{p})^{\lambda - \sum^{\lambda} x} \end{aligned} \quad (2)$$

Equation 2 is the *likelihood* of a Bernoulli distribution for switch i generating the best performing λ samples from a distribution with probability parameter p_i^{g+1}. Remember, $s^g_{\lambda:C,i}$ refers to the top λ (out of total C) performing switch configurations for switch i at generation g. The final expression of Equation 2 can be differentiated to find the value of $\hat{p} = p_i^{g+1}$ that maximizes this likelihood. We first apply a logarithmic transformation on this equation before differentiation, as shown below:

$$\ln(f) = \ln(\hat{p}) \cdot \sum^{\lambda} x + \ln(1-\hat{p}) \cdot (\lambda - \sum^{\lambda} x)$$

$$\frac{d\left(\ln(f)\right)}{d\hat{p}} = \frac{\sum^{\lambda} x}{\hat{p}} - \frac{\lambda - \sum^{\lambda} x}{1-\hat{p}} = 0$$

$$\sum^{\lambda} x - \hat{p} \cdot \sum^{\lambda} x = \lambda \cdot \hat{p} - \hat{p} \cdot \sum^{\lambda} x$$

$$\hat{p} = \frac{\sum^{\lambda} x}{\lambda} \quad (3)$$

Hence, the updated probability parameter $\hat{p} = p_i^{g+1}$ for each switch s_i's Bernoulli distribution B_i can simply be written as an average over the λ best performing candidates in generation g. We formalize this in Equation 4 below:

$$\hat{p} = p_i^{g+1} = \frac{\sum_{k=1}^{\lambda} s^g_{k:C,i}}{\lambda}. \quad (4)$$

Finally, in Figure 4, we first show the high-level representation of a single iteration of the MLE algorithm. Given a set of probabilities associated with the distributions B^g, we generate H_C^g samples of possible NoC configurations. We then select the top-K best-performing solutions $H_{\lambda:C}^g$ to revise the distributions as B^{g+1} for the next iteration. We also represent an actual trace of probability evolution across a complete MLE evolution for a 2×2 NoC in Figure 5. We note that the first 20 iterations show the results of a sampling when the $p_i \approx 0.5$ for the switches. With subsequent iterations learning from the combinations that worked well in the first phase to converge towards the best-performing switch configurations for each round. While MLE can also be used for Bayesian inference [2]

with respect to how samples are generated and probabilities updated, for our scenario we directly minimize an objective function rather than performing a classification.

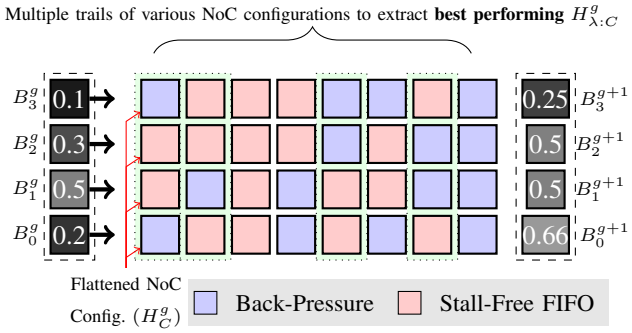


Fig. 4: Evolution of switch type probabilities for a 2×2 NoC using Maximizing Likelihood Estimation (MLE). On the **left, one MLE iteration is shown which has a candidate size C=8 (each column is a candidate) and elite size λ=4. Only bold NoC configurations are used to generate the new Bernoulli distribution B^{g+1}.**

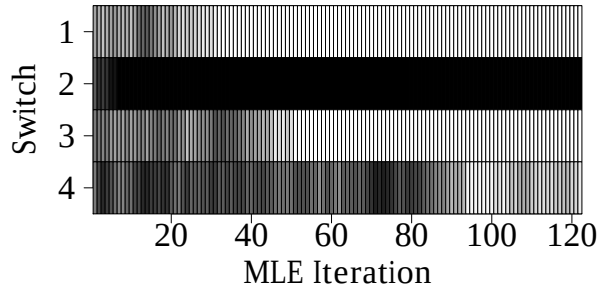


Fig. 5: The complete MLE algorithm is shown for a regulation rate of 0.5 and a block size of 16. Switch colors that settle for white are Backpressure switches while those that are Black are FIFO-based.

IV. Evaluation

A. Methodology

Our MLE optimizer is written in Python3. We implement the latency and buffer analysis in Matlab, and convert it to C code using Real Time Workshop; the optimizers communicate with the analysis tool using direct data transfer based on the Python `ctypes` APIs. We run all experiments on a 16-core Intel Xeon E5-2697A CPU and parallelize our search across 32 threads. We measure our algorithm to be 50-500× faster when compared against open-source Python implementations of black-box optimizers `CMA-ES` [6] and `RBFOPT` [3], while exploring the solution space just as effectively.

We evaluate our framework across a range of 100 synthetically-generated communication workloads with `RANDOM` and `LOCAL` communication patterns. For `RANDOM` patterns, each PE chooses destinations via uniform sampling of other PEs. For `LOCAL` patterns, sampling is restricted to a +/- 2 radius of neighbouring PEs. We run our analysis across various data block sizes to mimic the diversity of communication interfaces and endpoints like DRAM, PCIe, and Ethernet. Furthermore, we also test for optimizing real-world applications designs for FPGA accelerators such as Sparse matrix vector multiplication (`SpMV` [1]) used by many deep learning kernels [17] and `Graph` [14] analytics. The extracted traces exhibit a rich diversity of bandwidth requirements; with load factors of 37-92% between the benchmarks.

In addition to the static analysis above, we also implement and verify the entire NoC on a Pynq-Z1 board, with unified HopliteBuf+BP switches in the FPGA fabric (PL), and configuration tools for the NoC switches and regulators running on the ARM A9 (PS). For the unified NoC, the software layer on the ARM configures the switches *dynamically* over an AXI bus, depending on the application requirements, without the need for complete bitstream reconfiguration. For a 6x6 network, the runtime of the MLE optimization on the PS is 9-60s, and the reconfiguration of the switches instantiated in the PL takes 850ms.

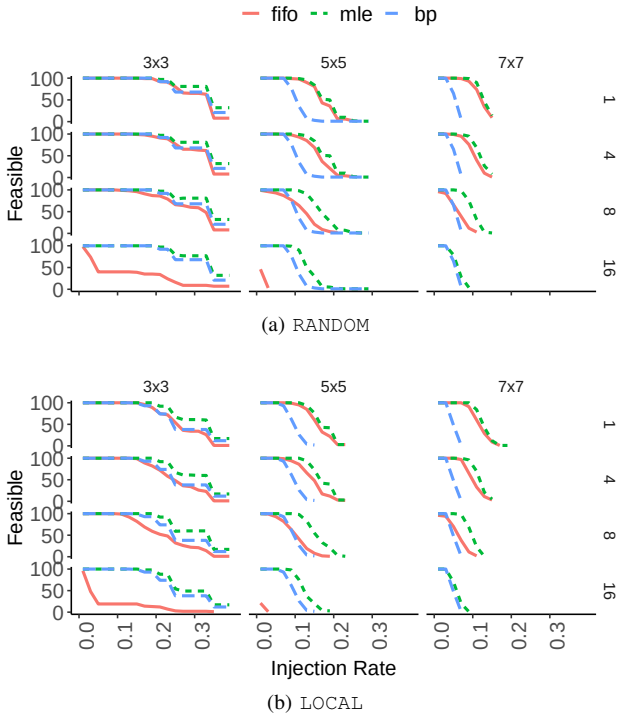
B. Results

In this section, we present the results of the static analysis of our hybrid FPGA NoC under various system sizes, data block sizes, and optimization methods. We are primarily interested in determining network feasibility (bounded latency), FIFO size needed, worst-case latency, fitness of the solution, and time required to optimize the NoC. We use the HopliteBP (bp) and HopliteBuf (fifo) cases as baselines where all switches in the NoC are configured one way or another.

1) Learning for Feasibility: In Figure 6, we determine the subset of the 100 synthetic `RANDOM` and `LOCAL` flowsets that can be routed feasibly on the NoC, that is, with bounded latencies and fit within the 32-deep FIFO capacity limits of an SRL32 structure. We configure MLE to produce feasible NoC configurations for provided flowset. As we vary system size, regulation rate, and data block size, we note several interesting trends:

- First, we observe that feasibility rate drops from 100% to 0% as we increase regulation rate (x-axis), with steeper losses observed for larger system sizes. This is understood as there are more flows competing for bandwidth that does not scale linearly with system size. For a torus, doubling of system size increases bisection bandwidth by only $\sqrt{2}$.
- As we increase data block lengths from 1–16 (figures along a column), we note a counter-intuitive effect. Now the HopliteBP and MLE designs show the highest feasibility envelopes while HopliteBuf loses severely. HopliteBuf is unable to maintain feasibility as the SRL32 FIFOs run out of capacity for larger data blocks. It is possible to increase feasibility by increasing FIFO sizes, but that impacts LUT cost and causes HopliteBuf to exceed the footprint of the HopliteBP switches.

- As we increase system size, we note that HopliteBP designs start to lose feasibility quickly due to pessimism in the analysis for backpressure-based designs. For pipelined backpressure-based switches, the analysis must account for the cascading effect of network flows, which significantly depresses sustainable rates. `LOCAL` pattern suffers a greater loss in feasibility due to the short distances traversed by the flows and resulting larger conflict set of flows.
- Finally, we observe that MLE-optimized NoC smoothly navigates the entire design space to deliver the highest feasibility rates across all combinations. This confirms that neither extreme solution works best in all cases, and a mix-and-match approach is necessary to get the best outcomes. Importantly, by mixing and matching switch configurations, MLE NoCs exceed the potential of either NoCs in isolation, achieving ≈2–3× higher feasibility over vanilla designs.



(a) `RANDOM`

(b) `LOCAL`

Fig. 6: Feasible set of `RANDOM` and `LOCAL` flowsets on system sizes 3×3–7×7 (*columns*) and data block sizes 1–16 (*rows*)

2) Learning for Worst-Case Latency: Next, in Figure 7, we show the effect of varying regulation rate of a 5×5 NoC when routing feasible `LOCAL` traffic traces. We configure the MLE to learn switch configurations to minimise `wclatency`. We summarise our observations with varying regulation rates and block sizes:

- First, we note that as the rate increases and reaches an inflection point, we see an increased spread in latencies across all configurations. This is due to the traffic regulation model dominating cycle counts below this inflection rate, forcing the network congestion effects to take a backseat. Second, as we increase data block sizes, the latencies increase and the spread shift upwards. We also note that there is noise around the calculation of mean curves due to the shifting feasibility combinations of the flowsets.

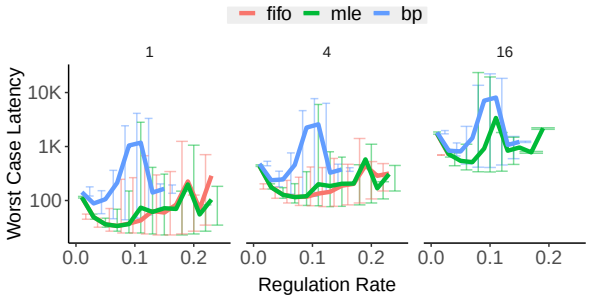
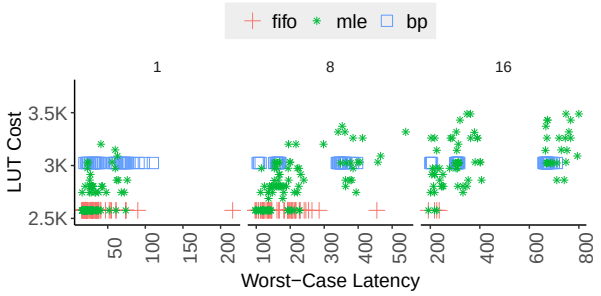


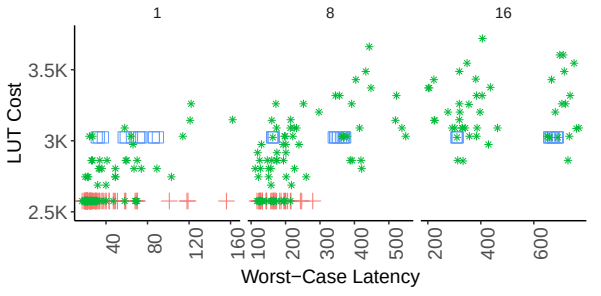
Fig. 7: Worst-case latency as a function of regulation rate for 100 synthetic `LOCAL` flowsets for various data block sizes.

- HopliteBuf NoCs are competitive at lower regulation rates. At higher regulation rates, the worst-case latencies can be quite large for certain flowsets. This is a direct result of deeper FIFOs and associated head-of-line blocking effects [12] for those scenarios. Also, HopliteBuf NoCs do not scale to large block sizes (=16) because they run out of stall-free FIFO capacity.
- The HopliteBP NoCs initially start with higher latency at the lower rates, but become competitive at larger regulation rates. At higher rates, HopliteBP NoCs suffer from loss of feasibility and cease to scale. At higher data block sizes, HopliteBP NoCs continue to scale beyond HopliteBuf for higher rates but saturate below MLE.
- MLE-optimized NoCs deliver competitive latencies across all rates and block sizes. At lower rates, the MLE-optimizer prefers reducing FIFO size and mimics HopliteBuf solutions as the regulation dominated latency is a fixed effect. At increased rates, MLE can strategically replace congestion hotspots with HopliteBP designs and avoid the increased buffering effects that cause high latencies for HopliteBuf designs. At large data block sizes and rates, MLE-optimized NoCs are the only feasible combinations outperforming **both** HopliteBuf and HopliteBP NoCs.

3) Learning for Cost Constrained Worst-Case Latency: For the next set of experiments, we configure the MLE to optimise for the cost constrained latency function of `wclatency*cost`, with an aim to analyse its ability to balance the growth of the two negatively correlated terms while simultaneously minimising their product. In Figure 8, we look at cost-latency tradeoffs when considering feasible `RANDOM` and `LOCAL` flowsets that are routed at a system size of 4×4 and a regulation rate of 0.17. At small data block sizes, MLE-optimized NoCs end up occupying the cost range closer to the lower-cost HopliteBuf designs. As block sizes increase, vanilla FIFO designs run out of capacity and declare infeasibility and MLE-optimized NoCs consume increasingly more LUTs. This suggests that MLE will replace HopliteBuf switches with the more expensive HopliteBP/HopliteBP+Buf versions in exchange for feasibility or proportionate latency gains. We also note the narrower spread of HopliteBP latencies



(a) RANDOM

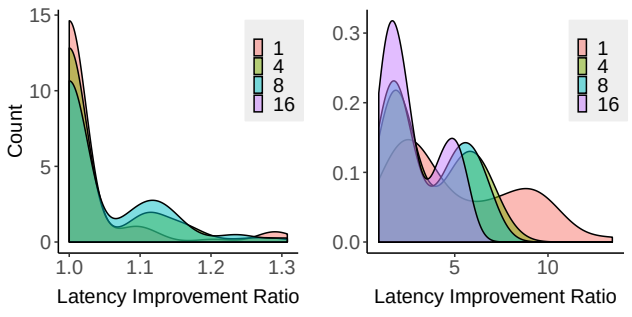


(b) LOCAL

Fig. 8: LUT Cost-Worst Case Latency tradeoffs for a 4×4 NoC with 0.17 regulation rate for 100 synthetic RANDOM and LOCAL flowsets across various block sizes.

which is a direct result of having fewer feasible combinations at that rate and block size.

We also quantify the extent of latency improvement distribution over HopliteBuf and HopliteBP switches in Figure 9 for 100 LOCAL flowsets at 0.13 regulation rate across different data block sizes on a 4×4 NoC. When compared to HopliteBuf designs, we note latency reduction by as much as 2× with little sensitivity to block sizes. When compared against HopliteBP, the latency wins can be as much as 15×. This larger win is attributed to pipelining effects in backpressure-based networks that can cause the analysis to include a large number of flows in the conflict set to produce safe upper latency bounds.



(a) vs. HopliteBuf (b) vs. HopliteBP

Fig. 9: Cumulative distribution of improvements of worst-case latency MLE NoC over HopliteBuf and HopliteBP.

4) Routing Real World FPGA Applications: We now show the effect or regulation on worst-case latency of realistic FPGA workloads running on a 16-client NoC. In Figure 10, we study the effect of regulation rate on worst-case latency of flows with a data block size of 4. With low congestion at low rates, all NoC designs exhibit similar characteristics. As we increase injection rates, we notice increased worst-case latencies across most workloads. In particular, we note greater feasibility and lower worst-case latencies for MLE-optimized NoCs. For smaller data block sizes (See Table II), MLE prefers HopliteBuf NoCs as most conflicts can be absorbed in the shallow SRL FIFOs. However, smaller blocks sizes require extremely wide NoCs with hundreds of bits of payload sent as a single block. We conclude that MLE-optimized hybrid NoCs are at par or better than either designs by sustaining 1-9× higher rates while achieving 1-6.8× lower latency across all real application benchmarks and block sizes.

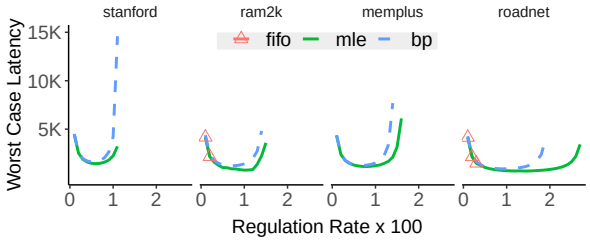


Fig. 10: Latency Scaling of a 4x4 NoC for Graph/ SpMV traces

		Rate×10^3			*Latency*				*Rate*×10^3			*Latency*		
		F	B	M	F	B	M		F	B	M	F	B	M
wiki	1	17	8	17	743	6.7K	743	google	1	17	8	17	789	6.7K	789
	4	–	8	8	–	16.2K	16.2K		4	–	8	8	–	16.1K	16.1K
stnfd	1	23	11	23	1310	6K	1310	soc	1	17	8	17	789	8.5K	789
	4	–	11	11	–	14.5K	3.2K		4	–	8	8	–	20.2K	20.2K
dac	1	19	10	19	1238	10.6K	1216	rdnet	1	37	19	37	1661	1609	1635
	4	–	10	10	–	25.6K	25.6K		4	3	19	27	1.4K	3.9K	3.4K
ram2k	1	28	14	28	998	2K	998	mmpls	1	27	14	27	2K	3.2K	2K
	4	2	14	15	2.1K	4.8K	3.5K		4	–	14	16	–	7.6K	6.1K
gene2	1	17	8	17	789	11.3K	789	bmhf3	1	25	14	25	1.5K	1.9K	1.5K
	4	–	8	8	–	27K	27K		4	–	14	14	–	4.7K	4.6K
bmhf2	1	25	12	25	1.6K	10.5K	1.5K	bmhf1	1	20	9	20	720	5.2K	720
	4	–	12	12	–	26.2K	3.8K		4	–	9	9	–	12.4K	12.4K
amzon	1	17	8	17	789	11.3K	789	add20	1	19	9	19	898	5.2K	898
	4	–	8	8	–	27K	27K		4	–	9	9	–	12.4K	12.4K

TABLE II: Application Performance over block size of 1,4 for Graph/SpMV traces. (F: FIFO, B: Back-Pressure, M: MLE)

5) Analysing Efficiency of MLE: Finally, we turn our attention to the MLE optimization flow and try to understand how the solver discovers good solutions. In Figure 11, we plot the solution quality (LUT cost × worst-case latency) and time taken to discover the solutions across a range of 100 synthetic RANDOM workloads targeting a 5×5 NoC at a regulation rate of 0.1. The HopliteBuf and HopliteBP NoCs are one-shot solutions that do not need any search and are hence observed in the left corner of the space. MLE's highly composable binary decision making proceeds in an iterative

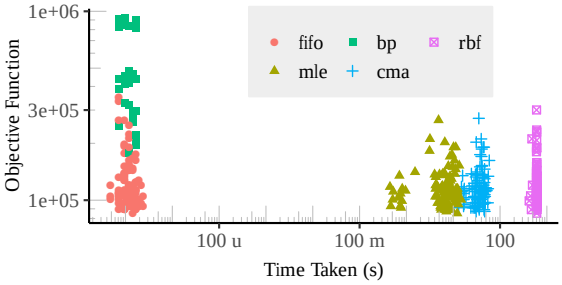


Fig. 11: Solution quality and time taken by MLE optimizer across flowsets mapped to a 5×5 NoC with 0.1 regulation rate.

fashion before it stabilizes. We observe a narrower spread of objective function values after 1–10 s. CMA-ES [6] explores the space just as effectively but takes ≈5–10× longer due to the necessary but obtuse integer quantization of real-valued distributions used by the optimizer. We also use RBFOPT [3], that generates marginally inferior solutions and is 50–500× slower than our MLE approach. While RBFOPT claims to require fewer explorations, the number crunching after each sample is compute intensive and dominates the fast analysis process for our problem.

V. Conclusions

In this paper, we show how to evolve hybrid FPGA NoC switch parameters to deliver a combination of feasibility, worst-case latency, and cost improvements over homogeneous FPGA NoCs. We demonstrate this by combining Hoplite-Buf, a stall-free FPGA NoC, with HopliteBP, a lightweight backpressure-based FPGA NoC using a fine-grained per-switch static configuration model. We use Maximum Likelihood Estimation technique to evolve NoC configurations that offer ≈2-3× improvements in feasibility, ≈1–6.8× in worst-case latency over synthetic and real world applications.

Source Code: https://git.uwaterloo.ca/watcag-public/hoplite-ml

References

[1] R. F. Boisvert, R. Pozo, K. Remington, R. F. Barrett, and J. J. Dongarra. Matrix market: a web resource for test matrix collections. In *Quality of Numerical Software*, pages 125–137. Springer, 1997.

[2] B. P. Carlin and T. A. Louis. *Bayes and empirical Bayes methods for data analysis*. Chapman and Hall/CRC, 2010.

[3] A. Costa and G. Nannicini. Rbfopt: an open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation*, 10(4):597–629, Dec 2018.

[4] T. Garg, S. Wasly, R. Pellizzoni, and N. Kapre. Hoplitebuf: Fpga nocs with provably stall-free fifos. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, pages 222–231, New York, NY, USA, 2019. ACM.

[5] T. Garg, S. Wasly, R. Pellizzoni, and N. Kapre. Hoplitebuf: Network calculus-based design of fpga nocs with provably stall-free fifos. *ACM Trans. Reconfigurable Technol. Syst.*, 13(2), Feb. 2020.

[6] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[7] Y. Huan and A. DeHon. FPGA optimized packet-switched NoC using split and merge primitives. In *Field-Programmable Technology*, pages 47–52, Dec. 2012.

[8] S. Jeon, J. Cho, Y. Jung, S. Park, and T. Han. Automotive hardware development according to iso 26262. In *13th International Conference on Advanced Communication Technology (ICACT2011)*, pages 588–592, Feb 2011.

[9] N. Kapre and J. Gray. Hoplite: Building austere overlay nocs for fpgas. In *Field Programmable Logic and Applications*, pages 1–8, Sept 2015.

[10] N. Kapre and J. Gray. Hoplite: A deflection-routed directional torus noc for fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 10(2):14:1–14:24, Mar. 2017.

[11] N. Kapre, H. Ng, K. Teo, and J. Naude. Intime: A machine learning approach for efficient selection of fpga cad tool parameters. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 23–26, New York, NY, USA, 2015. ACM.

[12] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, 35(12):1347–1356, 1987.

[13] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, 2001.

[14] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

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

[16] M. K. Papamichael and J. C. Hoe. Connect: re-examining conventional wisdom for designing nocs in the context of fpgas. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pages 37–46. ACM, 2012.

[17] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, 2017.

[18] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel. Network-on-chip programmable platform in versaltm acap architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, pages 212–221, New York, NY, USA, 2019. ACM.

[19] S. Wasly, R. Pellizzoni, and N. Kapre. HopliteRT: An efficient FPGA NoC for real-time applications. In *F. Program. Technol. (ICFPT), 2017 Int. Conf.*, pages 64–71. IEEE, 2017.