HopliteML: Evolving application customized FPGA NoCs with adaptable routers and regulators

GURSHAANT MALIK, University of Waterloo, Canada
IAN ELMORE LANG, University of Waterloo, Canada
RODOLFO PELLIZZONI, University of Waterloo, Canada
NACHIKET KAPRE, University of Waterloo, Canada

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 and workload adapted regulation. 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 through 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. We can further improve a workload's routability by learning to surgically tune regulation rates for each traffic trace to maximise available routing bandwidth. We capture critical dependency between traces by modelling the regulation space as a multivariate gaussian distribution and learn the distribution's parameters using Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We also propose *nested* learning that learns switch configurations and regulation rates in-tandem. Compared to standalone switch learning, this symbiotic nested learning helps achieve ≈1.5× lower cost constrained latency, ≈3.1× faster individual rates and ≈1.4× faster mean rates. We also evaluate improvements to vanilla NoCs' routing using only standalone rate learning (no switch learning); with ≈1.6× lower latency across synthetic and real world benchmarks.

CCS Concepts: • **Computing methodologies → Machine learning**; • **Networks → Network on chip**; • **Hardware → Reconfigurable logic and FPGAs**.

ACM Reference Format:
Gurshaant Malik, Ian Elmore Lang, Rodolfo Pellizzoni, and Nachiket Kapre. 2021. HopliteML: Evolving application customized FPGA NoCs with adaptable routers and regulators. *ACM Trans. Reconfig. Technol. Syst.* 1, 1, Article 1 (January 2021), 34 pages. https://doi.org/10.1145/3507699

Authors' addresses: Gurshaant Malik, gsmalik@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada; Ian Elmore Lang, ielmorlang@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada; Rodolfo Pellizzoni, rpellizz@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada; Nachiket Kapre, nachiket@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada.

1936-7406/2021/1-ART1 $15.00
https://doi.org/10.1145/3507699

1 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 [25] 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 [19], 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 [26], 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.

An application flowset is composed of a multitude of unique traffic traces (or flows), each carrying packets between a source and destination PE. Each flow is characterised by its regulated injection rate (the rate at which packets can be injected into the NoC) and data block size (number of consecutive packets injected). We explain the composition of a flow in greater detail in section 4.1. 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 Fig. 2a, as HopliteBP suffers from backpressure propagated on the horizontal connections. However, for vertical traffic in Fig. 2b, HopliteBP delivers 1.5× better worst-case latency. The cyclic loop of dependencies between flows shown in Fig. 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 3. 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 given QoR (quality of result) function. We formulate a Maximum Likelihood Estimation solution where we adjust the probabilities of each switch configuration based on iterative analysis trials.

NoCs make use of regulators to guarantee application bandwidth SLAs by mitigating a variety of packet starvation, deadlocks and denial of service scenarios. For a given application flowset consisting of variety of traffic traces, HopliteBP and HopliteBuf rely on token bucket regulators [15] for every trace to schedule the injection of its packets into the NoC. For a given flow, its token bucket regulator limits the injection rate and data block size of that flow to its prescribed regulation rate (ρ) and burst size (b) respectively. We explain the token bucket regulator in greater detail in section 4.2. Regulation is an important variable in the Hoplite suits of analysis tools to guarantee tight bounds on routing latency and NoC costs. Previous versions of Hoplite [5, 17] leave regulation rate out of the decision space as a user parameter; evaluating NoC designs by measuring performance over a range between [0,1] with a fixed increment linear search. Furthermore, only a single regulation rate is used to regulate all traces in an application flowset, which can create unintended bottlenecks in NoC routing and distort search spaces of other NoC parameters. This paper extends the scope of regulation by *learning* regulation rates for *each trace*, with the aim of maximising the requested QoR, specific to the given application flowset. This allows us to surgically address localised contention bottlenecks within the NoC by individually regulating traces involved in the bottlenecked hotspot. We encode the rates for a flowset as a multivariate gaussian distribution to capture the dependence of a trace's routability on other traces within the flowset and learn the parameters of this distribution using Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [6]. Furthermore, we introduce nested learning where we learn the switch configurations of the NoC and the regulation rates of traces *in tandem*. This coupling of two search spaces allows one to modulate the other in a synergistic manner, thus allowing for a more effective NoC design that outperforms designs born out of learning only either one of the search spaces.

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.
- Development of a multivariate gaussian distribution based evolutionary learning strategy to learn regulation rates, specific to the application flowset and QoR requested.
- Development of a nested learning strategy to learn switch level configurations and regulation rates for an application flowset in tandem.
- Formalization of the Hoplite NoC analysis suite and explanation of its flow model; used for estimating feasibility and upper bounds on routing latencies, including worst case and injection latency.
- Quantification of worst-case latency, feasibility, cost and optimization runtime across real and synthetic applications. Demonstrating:
  - ≈1–6.8× reduction in worst-case latency and ≈2-3× improved feasibility by learning switch level configuration.
  - When learning rates and switch configurations in-tandem (*nested learning*), a further ≈1.5× lowered cost constrained latency, ≈3.1× faster individual rates and ≈1.4× faster mean rates.
- We implement and verify our entire switch learning 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.

2 Switch Design

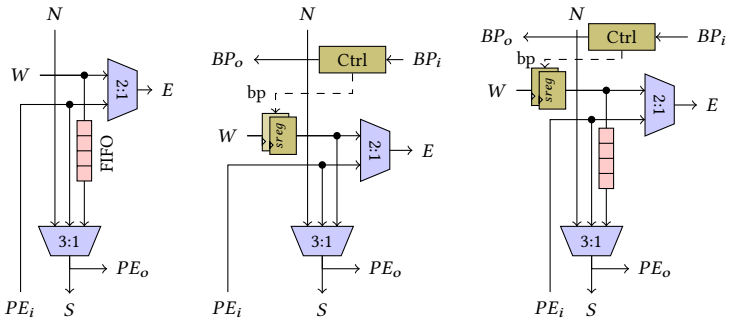
In this section, we first discuss the original Hoplite NoC [9] and related work around designing NoCs with timing predictability [13, 14, 21, 23, 28]. We then introduce the designs of the existing HopliteBuf and HopliteBP networks and details of their FPGA implementation. We focus on 3 properties specifically: 1. Tight latency bounds; 2. Lightweight area overheads and 3. Application awareness. 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 ($N \rightarrow S$). While the lightweight hoplite switches are an attractive proposition for area sensitive applications, possibility of routing livelock and unbounded routing latencies are significant limitations, making it unsuitable for systems requiring guarantees on worst-case packet latencies. Furthermore, this original version of the hoplite NoC cannot be tuned in an application aware manner; with the aim of optimising its routing performance by taking into account the application's traffic traces and their interactions with each other. Motivation for application aware tuning is presented in subsequent section 3.

[14] aims to provide latency guarantees by leveraging the use of probabilistic weighted arbiters and buffers along router ports to provide bandwidth equity to all traces on the NoC. However, global and local fairness delivered by these arbiter types comes at the expense of significant area and latency overheads for the NoC. Instead of providing fair arbitration, [23, 28] attempts to schedule traffic intro domain exclusive waves that can potentially travel unimpeded from other domain traffic through the NoC, thus providing strict isolation guarantees. However, contention between traffic belonging to the same domain is still possible. In such a scenario, losing packets have to wait for the next compatible domain wave to traverse the NoC, introducing unavoidable routing latency overheads and necessitating the use of buffers along router links. Further, independent traffic is not grouped into domains in an application aware manner, which can severely constrain the NoC's available injection bandwidth (since packets can only *surf* their own domain wave). [23]'s use of time division multiplexing at the virtual channel also results in router complexity growing linearly with virtual channel counts and attempts to offset this area overhead via explicitly pipelining routers limits the number of domains that can maintain strict flow isolation and zero delay hop routing [23, 28]. Leveraging TDM asynchronous routers in a GALS implementation, [13] introduces a simple router design. However, the TDM is statically scheduled and is not tuned to optimise the underlying application's QoR. The GALS implementation also mandates expensive modifications to the network interface and additional design cycles for clock domain crossing compatibility between the NoC and user PEs. Instead of equitable arbitration [14] or domain creation [23, 28], [21] guarantees conflict free traffic flow by leveraging dependency graphs between interacting traces to introduce strategic delays for each traffic trace. However, this forces even non conflicting traffic traces to be delayed by an amount determined by the dependency graphs. Furthermore, only a single packet can be inserted in a time-slot, increasing queuing latency and lowering available network throughput. These strategic delays are introduced by delay latches across the router paths, which also increases the area overhead of NoC. Finally, despite non interacting traffic traces, [21] still requires flow control to avoid end-point saturation. Thus, FPGA application design still needs a *lightweight* NoC that addresses this gap by providing *strict latency bounds* that are *optimised for the application* being routed.

2.1 HopliteBuf and HopliteBP

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. It is worth noting that while the FIFO is only available for $W \rightarrow S$ packets, we also need to hold back $PE \rightarrow S$ packets if a N packet is present, as the PE port has the lowest priority for injecting a packet; the same holds for a $PE \rightarrow W$ packet if a higher priority $W \rightarrow E$ is present. PE packets have lowest priority amongst all packets and must wait for traffic to clear on the target exit mux. Rather than implementing additional, LUT-expensive FIFOs, we assume a regulator compatible PE interface that can be stalled; such that the PE is prevented from injecting a packet in any clock cycle when a higher priority packet is present. We assume that packet delivery to a PE is not stallable, thus 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 (BP_0) upstream in the opposite direction of packet flow [18].



(a) HopliteBuf (FIFO) (b) HopliteBP (Backpressure) (c) **Unified** HopliteBuf+BP

Fig. 1. Block diagrams of HopliteBuf, HopliteBP and HopliteBuf+BP designs.

HopliteBuf+BP (FIFO + Backpressure in Fig. 1c) is a unified switch that includes both switch components and supports runtime configuration of the operating mode. The HopliteBuf design presented in [4] cannot coexist with HopliteBP switches in the same horizontal ring, as it will not respond to a backpressure signal (BP_0), and thus, despite assertion of this signal, may end up sending a packet to the W input of a backpressured HopliteBP switch. In order to have switches with a $W \rightarrow S$ FIFO in the same horizontal ring as HopliteBP switches, we define this third switch, that extends HopliteBuf by incorporating backpressure and shadow registers in the design.When implementing a soft NoC, a HopliteBuf+BP switch is used in place of HopliteBuf whenever the analysis determines that the switch could be reached by a backpressure signal generated by a HopliteBP switch on the same horizontal ring. We can also realize a hard, configurable NoC architecture based on the same principle. Specifically, the HopliteBuf+BP structure can be used to support both stall-free FIFO and backpressure designs, by simply implementing a programmable

register that can either be configured during compilation or in response to evolving traffic conditions when routing a workload (without requiring full FPGA bitstream reconfiguration).

All 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 32 bit leaky bucket regulator [15], which uses two parameters, regulation rate $\mathbf{f_i}.\boldsymbol{\rho}$ and data burst size $\mathbf{f_i}.\mathbf{b}$: data burst 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. We review the static analysis used in these designs in greater detail in Section 4.

2.2 Xilinx FPGA Mapping

The muxes of the original Hoplite switch [26] 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 1. 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 1. Resource utilization of Hoplite based switches on Xilinx Virtex-7 for 32b datawidth and 32-deep SRL32 FIFOs.

3 Motivating Example

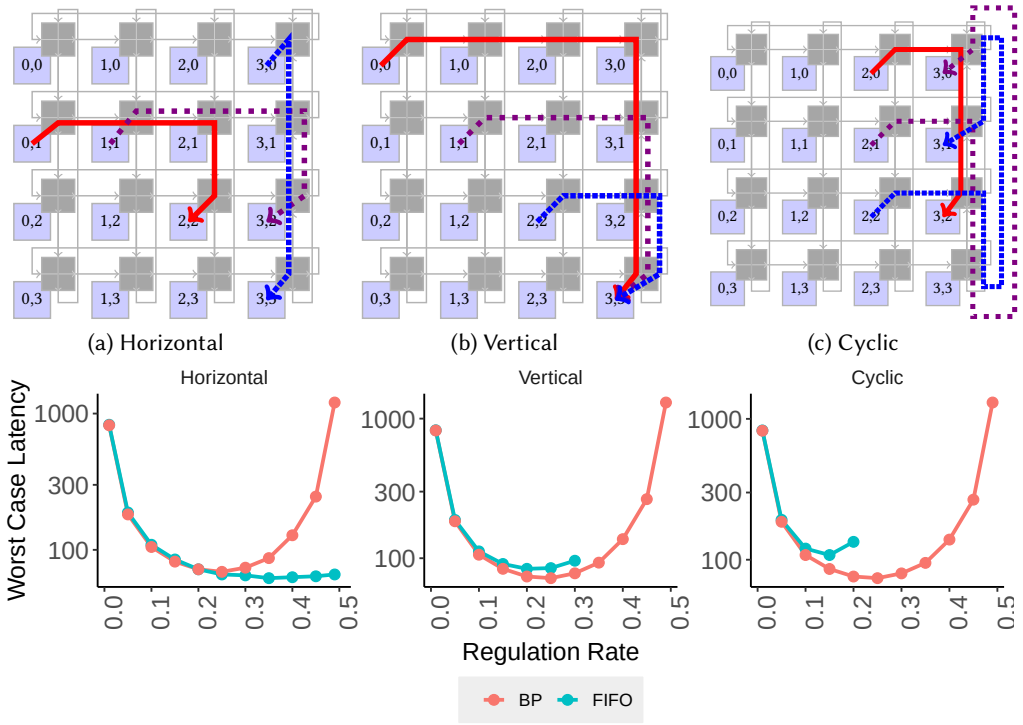
In this section, we discuss the motivation for learning NoC parameters for a given application flowset. Specifically, in order to maximise a user requested QoR (quality of result) for a specific application flowset, we build a case for:

(1) Building a hybrid NoC design by learning mode of operation (between HopliteBuf and HopliteBP), for every switch in the NoC.
(2) Learning the regulation rate for every traffic trace present in the application flowset.

3.1 Learning Switch Configuration

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

In Fig. 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

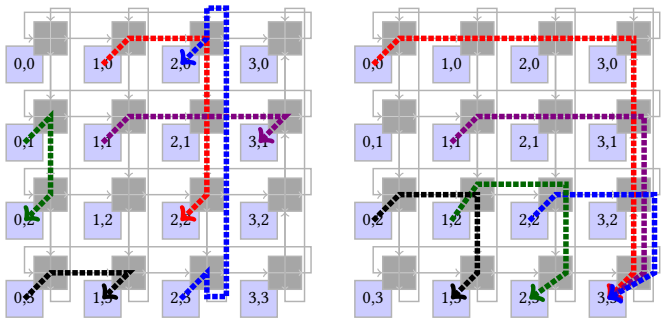


(d) Worst-Case Latency Scaling Trends

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

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).

- **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.
- **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) Localized Contention (b) Turning Traffic in vanilla Hoplite-Buf

Fig. 3. Interacting network flows on a 4×4 NoC arranged to show the benefits of learning regulation rates for each traffic trace

A user-supplied set of network flows can contain a combination of these three patterns, among others. The first 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.

3.2 Learning Regulation Rates

We now make a case for learning regulation rates for every trace in an application flowset by presenting a set of arguments using Fig. 3a and 3b. Specifically, each traffic trace within an application flowset interacts with other traces in a complex non-deterministic manner, thus affecting NoC routing. Flowset specific learning allows us to maximise NoC routability by encoding the latent affects of these traces' interactions into the learnt rates. We now discuss two specific advantages of learning unique regulation rates for every trace in the flowset, specific to the applications and QoR at hand. We contrast this proposed learning approach against the practice of using a single rate to regulate all traces within a flowset [5, 17].

- **Local contention zones**: In Fig. 3a, the NoC comprises of a localised contention hotspot around switch (2,1), comprising of 3 traces (▪▪▪, ▪▪▪, ▪▪▪). Regardless of mode of operation of the switches, these 3 traces generate more contention than the other two traces (▪▪▪, ▪▪▪) if we regulate all of them by the same amount. As a result, this localised hotspot forms a bottleneck in the feasibility and latency analysis, thus leaving available NoC bandwidth untapped by (▪▪▪, ▪▪▪). For example, the non conflicting traces' (▪▪▪, ▪▪▪) regulated injection rates can reach levels that are much higher than the hotspot's, without having any adverse affect on the routability of the application flowset. An application flowset can generate a multitude of these disparate contention zones within a NoC. Learning *unique trace level* regulation rates, specific to the application flowset, allows us to extract information about such trace interactions and recover lost NoC routing bandwidth, thus allowing for a more efficient optimization of the application QoR.
- **Learning switch configuration**: In Fig. 3b, we investigate learning switch level configurations for the traces shown. We start off with all of the NoC's switches operating in HopliteBuf mode. The application flowset consists of predominantly turning traffic traces. Note that

switch (3,2) south link has 3 traffic traces exiting through it (■■■, ■■■, ■■■) which forms a potential *choke point* that can quickly become infeasible as regulated injection rates of these three traces increase (FIFO runs out of capacity). The lost feasibility can be recovered by replacing this HopliteBuf switch (3,2) by a HopliteBP NoC. However, this can lead to frequent and incessant stall of its entire row due to backpressure signals induced from its west port. This unfairly penalizes innocent traces (■■■, ■■■) which were not even interacting with the infeasibility causing traces but now are made to sacrifice their available bandwidth to make routing feasible. Furthermore, the entire row must now be upgraded to a HopliteBP+Buf configuration to support backpressure and FIFO signals, resulting in a more costly NoC design. Instead, an easier alternative is to address the infeasibility at its source. We can tune the regulation rates of the traces most responsible for eating up FIFO capacity at switch (3,2); trace (■■■) since it is not only involved in the contention at switch (3,2) but also is responsible for increasing the data block size of the trace exiting at south port of switch (3,1).

Such subtle and non-deterministic interactions between traffic traces of a complex flowsets can be challenging for a NoC to address. The second key idea explored in this paper is to learn the regulation rates of each traffic trace in an application flowset to

(1) Surgically tune localised hotspots by encoding this contention information into regulation related decision making for affected traces.
(2) Increase efficiency of switch learning with nested learning where rate learning can help expand the domain of feasible switch solutions as well as directly improve NoC routing conditions.
(3) For a NoC of size NxN, efficiently navigate for solutions in a multi-modal search space that asymptotically grows as $(0, 1]^{N^4}$ and makes simple search based techniques impractical.

4 Latency and Backlog Analysis

We now discuss how to derive latency bounds, as well as backlog (the maximum FIFO occupancy) bounds, for a NoC design incorporating a mix of HopliteBuf and HopliteBP switches. Specifically, the analysis derives four sets of parameters:

- The worst-case injection latency for any flow f_i, that is, the maximum time that node $(f_i.xs, f_i.ys)$ can be stalled while trying to inject one or more packets of f_i.
- The worst-case queuing delay for any flow f_i turning $W \rightarrow S$ on a FIFO enabled (HopliteBuf/Unified) switch, that is, the maximum time that a packet of f_i can be queued in the FIFO.
- The per-hop delay for any flow f_i, from (xs, ys) to (xd, yd). The total latency for f_i is then the sum of its injection latency, queuing delay (if applicable), and per-hop delay.
- Finally, to prove that stall-free FIFOs can be used without dropping packets, we determine the worst-case backlog (buffer occupation) for any FIFO enabled switch.

We begin by introducing the flow and traffic regulation model. We then summarize the existing analysis for HopliteBuf [4] and HopliteBP [5], and finally show how to extend it to a hybrid architecture.

4.1 Flow Model

We assume that each NoC node sends packets as part of one or more flows; a flow f_i represents a sequence of data blocks, each comprising $f_i.k$ consecutive packets, sent between the same source node $(f_i.xs, f_i.ys)$ and destination node $(f_i.xd, f_i.yd)$. The data block size $f_i.k$ is related to the architecture of the PEs and the width of the NoC; as an example, if blocks represent 32 bytes cache lines and the NoC width is 8 bytes, we obtain $f_i.k = 4$. We make no assumption on the exact time

at which successive data blocks are produced by the source node, as such information might be too difficult to analyze [1]. Instead, our goal is to determine a worst-case analytical bound to the latency suffered by any data block of flow f_i. The bound includes both the injection latency $Injection(f_i)$, that is, the maximum time that node $(f_i.xs, f_i.ys)$ takes to inject all packets of the block into the network, as well as the queuing delay, that is, the maximum time that any packet of f_i can be queued in a $W \rightarrow S$ FIFO. As part of the queuing analysis, we also derive a backlog bound for each FIFO. We say that a network is stable if all flows have bounded total latency and the backlog for all FIFOs is smaller or equal to the FIFO size.

4.2 Traffic regulation

We are interested in computing a *composable* latency bound, that is, the bound for a flow should not depend on detailed information on the activity of unrelated PEs. To this end, we limit the maximum number of packets that a flow can inject in the network in any interval of time using a network *regulator*. In this way, to compute the latency for f_i we only need to know the regulation parameters and source/destination for every other flow f_j; as long as those remain constant, any change to the way f_j generates data blocks has no effect on f_i. As in [4, 5], we employ a token bucket regulator [15], which uses two parameters, regulation rate $f_i.\rho$ and burst size $f_i.b$: burst size 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.The implementation of the regulator uses two counters. The rate counter is incremented every cycle and adds a token to the token counter every $1/f_i.\rho$ cycles (the regulation period). The token counter has a maximum size of $f_i.b$. At any clock cycle, the node is allowed to inject a packet only if the NoC is ready (meaning that no other packet is being sent to the same output port, given that the PE port has the lowest priority) and the token counter is not empty; sending a packet removes a token from the counter. There is no buffer associated with the regulator; hence, a node that wishes to inject a packet but is blocked by the regulator is simply stalled. We make no assumption on the arbitration policy among flows sourced from the same node: if during a clock cycle a node is ready to inject a packet from both a flow f_i and a flow f_j, then it can select either of the two flows.

4.3 Injection Latency

To determine the injection latency for flow f_i, we need to compute the set of conflicting flows Γ_i^C that can prevent a packet of f_i from being injected in the NoC. This set comprises all flows sourced from the same node $(f_i.xs, f_i.ys)$ as f_i, plus all higher priority flows that traverse the switch at $(f_i.xs, f_i.ys)$ in the same direction as f_i: the $W \rightarrow E$ flows if f_i is injected to the E port, or the $N \rightarrow S$ and $W \rightarrow S$ flows if f_i is injected to the S port. After obtaining the conflict set Γ_i^C, the injection latency can be computed as follows, where $b(\Gamma_i^C)$ is the sum of data block sizes of all flows in Γ_i^C and $\rho(\Gamma_i^C)$ is the sum of their regulation rates:

Theorem 1 (Theorem 1 in [5] and 2 in [27]). *Assume $\rho(\Gamma_i^C) < 1$ and the client wishes to inject a sequence (data block) of $f_i.k \leq f_i.b$ packets for flow f_i. Then the delay to inject all packets in the block is upper bounded by:*

$$Injection(f_i) = \lceil 1/f_i.\rho \rceil - 1 + \left\lceil \frac{b(\Gamma_i^C)}{1-\rho(\Gamma_i^C)} \right\rceil + \left\lceil (f_i.k-1) \cdot \max\left(\frac{1}{f_i.\rho}, \frac{1}{1-\rho(\Gamma_i^C)}\right) \right\rceil. \quad (1)$$

[1]For example, in the case of requests produced by a cache, the specific timing of fetches/write-backs depends on both the initial cache state, and the path through the program.

Note that $1 - \rho(\Gamma_i^C)$ is the available regulation rate for f_i after removing the rate of flows in Γ_i^C. In essence, the first packet in the data block is delayed first by the regulation period (inverse of the rate) $1/f_i.\rho$; then, by the data block size of interfering flows, which is captured by term $b(\Gamma_i^C)/\big(1 - \rho(\Gamma_i^C)\big)$; and finally, the remaining $f_i.k - 1$ packets are injected at the minimum rate (maximum of the periods) between $f_i.\rho$ and the available regulation rate $1 - \rho(\Gamma_i^C)$. We make three important observations:

- If $\rho(\Gamma_i^C) \geq 1$, then in the worst case f_i can never inject, hence its latency is unbounded and the network is unstable.
- The bound only holds for data blocks of size at most $f_i.b$: larger blocks can be delayed by $b(\Gamma_i^C)$ multiple times, leading to significantly worse latency. Furthermore, increasing the burst size of the regulator beyond the flow's block size does not provide any advantage to f_i. While it could increase $b(\Gamma_j^C)$ for some other flow f_j, in the rest of the paper, we simply set $f_i.b = f_i.k$ for all flows.
- Increasing the regulation rate $f_i.\rho$ improves the latency of f_i, because it reduces the regulation period $1/f_i.\rho$; it also increases both the guaranteed $(1/Injection(f_i))$ as well as average-case regulation rate for f_i. However, it can also decrease the available rate $1 - \rho(\Gamma_j^C)$ for some other flow f_j, leading to a higher latency for f_j.

Therefore, the regulation latency of a flow represents a measure of how latency-critical the flow is: the higher the rate, the better the latency for that flow, at the expense of other flows. This competitive relationship between traces becomes important when learning unique rates to address inequity in per-trace regulation.

4.4 Analysis for FIFO Mode

For a flow f_i turning $W \rightarrow S$ on a FIFO enabled switch (vanilla HopliteBuf or Unified), let f_i' denote the flow after it leaves the S port: the effect of queuing packets in the buffer can cause the data block size $f_i'.b$ to be higher than $f_i.b$. In turn, this will affect the data block size and latency of flows either injected S or turning $W \rightarrow S$ on downstream switches in the same vertical ring. Hence, this effect must be bounded to obtain safe worst-case estimates. To this end, the analysis in [4] relies on the theory of network calculus [15], which is commonly employed to derive deterministic bounds on network latency and backlog. The analysis derives a set of linear equations that relate the data block size $f_i'.b$ of f_i' to the data block size of other flows leaving the S port on other FIFO switches on the same vertical torus. We then solve the system of equations; if resulting data block size values for all flows are finite and positive, then they must indeed represent valid upper bounds. Otherwise, we deem the network unstable. Assuming stability, the queuing delay for f_i and backlog for the switch can then be derived based on the regulation rate and data block size of all flows that traverse the switch in the $N \rightarrow S$ and $W \rightarrow S$ direction.

It is important to note that due to the torus topology, the system of equations can contain cyclic dependencies between data block size variables. An example is provided in Fig. 2c: here, each flow turning $W \rightarrow S$ is delayed by the other two flows that traverse the switch $N \rightarrow S$. Hence, increasing the data block size of a flow causes an increase in the data block size of the other two flows, which in turn affect the first one. This feedback loop introduces pessimism in the analysis, causing a reduction in the sustainable per-link utilization: for this example, the network becomes unstable at a per-link utilization of 75% (a regulation rate of 1/4 for each flow).

4.5 Analysis for Backpressure Mode

Since they do not contain a FIFO, no buffer analysis is required for backpressure enabled switches (vanilla HopliteBP or Unified). Instead, the analysis in [5] accounts for the effects of backpressure by adding all backpressured flows to the set of conflicting flows Γ_i^C. Specifically, if f_i turns $W \rightarrow S$ at a

switch, we add to Γ_i^C all other flows that traverse that switch in the $N \rightarrow S$ or $W \rightarrow S$ directions, as they can backpressure f_i. Furthermore, if another flow f_j shares a W port on any switch with f_i, then all flows that backpressure f_i can also indirectly cause backpressure on f_j; therefore, they must be added to Γ_j^C.

4.6 Hybrid NoC Analysis

To analyze our hybrid NoC, composed of individual switches operating in either FIFO or backpressure mode, we combine the analysis for FIFO and backpressure modes; we first run the buffer analysis for each vertical ring that contains at least one FIFO enabled switch. Note that for any flow f_i that turns $W \rightarrow S$ on that vertical ring at a backpressure switch, we consider the original data block size $f_i.b$ rather than the modified data block size $f_i'.b$, as the flow does not traverse a buffer. We then run the backpressure analysis for each horizontal ring that contains at least one backpressure switch. Note that a flow f_i can suffer backpressure from $N \rightarrow S$ and $W \rightarrow S$ flows only if it turns $W \rightarrow S$ at a backpressure enabled switch; however, a flow f_j can suffer indirect backpressure from f_i if it shared a W port on any switch (HopliteBP/Unified) or the horizontal ring. Finally, it is important to note that based on the analysis, adding a $W \rightarrow S$ FIFO buffer to a switch is a trade-off: on the negative side, the buffer increases the data block size of turning flows, and thus the interference on the vertical ring; on the positive, if we can prove that the FIFO is stall-free, we avoid generating backpressure on the horizontal ring. However, if the FIFO can be stalled, then it is not advantageous. For this reason, the HopliteBP design does not include a FIFO.

5 Evolutionary Learning of NoC Parameters

We now present the key ideas explored in this paper. We learn, for a given application flowset, data block size and defined QoR:

(1) The configuration mode (between HopliteBP and HopliteBuf) for every individual switch in the NoC.
(2) Unique regulation rates for every trace in the application flowset.

This allows us to tailor the NoC towards the application at hand by optimizing for a user defined QoR. We cast the learning of switch level configurations as a Maximum Likelihood Estimation problem. We learn regulated injection rates of every trace in the flowset using Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). We also present *nested* learning, where we learn both regulation rates and switch configurations in-tandem.

5.1 Learning NoC Switches

The first 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 Fig. 4, 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 (provided by user or a learning algorithm) and data block size of each flow, we determine whether each switch in the network implements

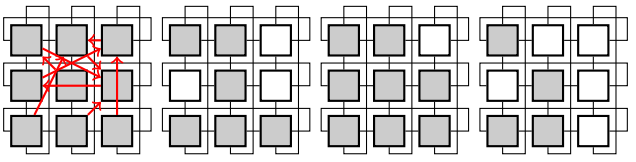


Fig. 4. FPGA NoC Switch Configuration for a 3×3 NoC for a set of nine flows (shown in leftmost subfigure) with identical but four combinations of rate ρ and block size b characteristics. At low rates and block sizes, most switches tend to be cheap HopliteBuf ▣ variants, while a growing number convert to expensive HopliteBP □ variants with increasing rate+size.

a stall-free FIFO or backpressure logic to optimise 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 $p_i = 0.5\ \forall$ 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 2. For our experiments we chose C=100.

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

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:

$$
\begin{aligned}
f\left(s^g_{1:C,i},\ldots s^g_{\lambda:C,i}|p_i^{g+1}\right) &= P\left(s^g_{1:C,i}, s^g_{2:C,i}\ldots, s^g_{\lambda:C,i}|p_i^{g+1}\right) \\
&= \underbrace{P(x_1, x_2\ldots, x_\lambda|\hat{p})}_{\text{substitute}} \\
&= \hat{p}^{x_1}\cdot(1-\hat{p})^{1-x_1}\ldots\hat{p}^{x_\lambda}\cdot(1-\hat{p})^{1-x_\lambda} \\
&= \hat{p}^{\sum^{\lambda} x}\cdot(1-\hat{p})^{\lambda-\sum^{\lambda} x} \qquad (3)
\end{aligned}
$$

Equation 3 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 3 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:

$$
\begin{aligned}
\ln(f) &= \ln(\hat{p})\cdot\sum^{\lambda} x + \ln(1-\hat{p})\cdot(\lambda-\sum^{\lambda} x) \\
\frac{d\left(\ln(f)\right)}{d\hat{p}} &= \frac{\sum^{\lambda} x}{\hat{p}} - \frac{\lambda-\sum^{\lambda} x}{1-\hat{p}} = 0 \\
\sum^{\lambda} x - \hat{p}\cdot\sum^{\lambda} x &= \lambda\cdot\hat{p} - \hat{p}\cdot\sum^{\lambda} x \\
\hat{p} &= \frac{\sum^{\lambda} x}{\lambda} \qquad (4)
\end{aligned}
$$

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 5 below:

$$
\hat{p} = p_i^{g+1} = \frac{\sum_{k=1}^{\lambda} s^g_{k:C,i}}{\lambda}. \qquad (5)
$$

Finally, in Fig. 5, 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 $H^g_{\lambda:C}$ to revise the distributions as B^{g+1} for the next iteration. 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.

5.2 Learning Regulated Injection Rates

The second key idea explored in this paper is to learn the degree of regulation introduced in each trace for a given application flowset. An application flowset can be composed of a diverse types of traces and much like the *butterfly effect*, each trace generates varying degrees of contention within the NoC via second order interactions with other traces. We wish to improve the overall routing conditions of these complex traffic interactions by regulating the amount of traffic generated by each trace within a time frame. Specifically, for a given application flowset, we learn these regulated injection rates, *for every trace in the flowset*, to maximise a user requested QoR function. We explore 2 methods of rate learning: 1. `Univariate` and 2. `Multivariate`.

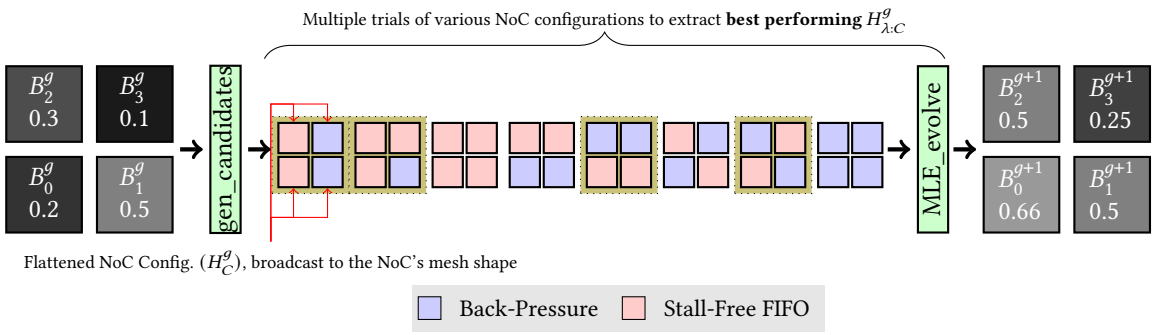


Fig. 5. Evolution of switch type probabilities for a 2×2 NoC using Maximizing Likelihood Estimation (MLE). One MLE iteration is shown which has a candidate size C=8 (each of the 8 shown 2x2 Hoplite NoC configurations is a candidate) and elite size λ=4. Only marked NoC configurations are used to generate the new Bernoulli distribution B^{g+1}.

5.2.1 *Univariate Rate Learning*

One method is to linearly search over a uniform distribution ∈ (0,1] with the increment precision truncated to 32 bits to mimic the regulation dynamics of the token bucket regulators. Each search sample is then universally applied to all traces' regulators and the resulting QoR determined. The sample that best optimises for the requested QoR can be selected as the desired regulation rate. This is equivalent to the linear search of [17] and section 6.2.1 (learning only switch configurations) ∈ (0,1], with NoC routability measured as a function of the regulation rate. The precision of the linear search increments can be fine-grained to support 32 bit regulators instead of the chunkier increments of the original work. Furthermore, the brute force linear search can be replaced with an intelligent binary search or a `univariate` gaussian modelling, with no effect to the final quality of solution. While information about the application flowset is encoded in this *light and fast* methodology, it fails to take into account any unique dependencies between traces and their latent effects on other regions of the NoC since all traces are equally regulated. Furthermore, it prevents the NoC from reaching its peak available bandwidth since regulation is bottlenecked by the worst trace in the flowset. We compare and contrast this `univariate` learning against `multivariate learning` in more detail in section 6.2.

5.2.2 *Multivariate Rate Learning*

We now present the main idea of learning regulation rates: modelling each trace's regulation rate as a `multivariate` gaussian distribution and learning this distribution's parameters using Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES) [6]. Formally, as shown in equation 6, for a NoC of size N×N, given a flowset F composed of set of traces $(f_1, f_2, ..., f_i, ..., f_n)$ and data block size ρ, we wish to learn the corresponding regulation rates R=$(r^{f_1}, r^{f_2}, ..., r^{f_i}, ..., r^{f_n})$ such that the user defined objective QoR function Q, calculated as a function of the Hoplite analysis suite H, can be maximised (or minimised).

$$F = (f_1, f_2, ..., f_i, ..., f_n) : n \leq N^4$$

$$R = (r^{f_1}, r^{f_2}, ..., r^{f_i}, ..., r^{f_n}) : r^{f_i} \in (0, 1] \forall i$$

$$R = \text{argmax}(Q(H(F, \rho, R))) \tag{6}$$

Recall that each trace in a flowset can impact the others via second order interactions and we encode this dependence by modelling R as a multivariate gaussian distribution $R \sim \mathcal{N}(\mu, \sigma^2)$, with $\mu \in [0,1]^n$ and $\sigma \in [0,1]^{nxn}$. This is a key differentiation between `univariate` of section 5.2.1 and [17] as it allows us to extract crucial information about these interactions. Furthermore, increasing the dimension of the regulation search space allows us to directly address bottlenecks in NoC routing by: 1. learning the regulation rates of the most difficult traces in an application as well as 2. encode this information in rates of traces that might be involved in indirect interactions. `Multivariate` learning, when paired with algorithms that learn other NoC parameters (switch configuration in section 5.3), also provides access to a much richer search space in contrast, when compared to `univariate` learning. This is explained by `univariate` unilaterally regulating all traces with the same amount, thus introducing a strong bias in other algorithm's search space.

We now explain the modelling of `multivariate` learning as a black box optimization problem using CMA-ES. Just like switch learning of MLE, the core of `multivariate` learning involves sampling C candidate rates $R_C^g \in (0,1]^{n^2 xC}$ from a multivariate distribution $\mathcal{N}(\mu_g, \sigma_g^2)$ at every generation g as shown in Equation 7. For our experiments, we choose $C = 100$. Each sample in R_C^g is then tested for fitness on the QoR function. As shown in Equation 8,the top λ performing regulation rate samples $R_{\lambda:C}^g \in (0,1]^{n^2 x\lambda}$ are then selected for CMA-ES to evolve the multivariate distribution in the general direction of top performing samples by updating the mean μ and covariance σ according to the evolutionary steps detailed in [6]. Observe in Equation 8 that CMA-ES maintains an evolutionary path to keep a historical track of past generation's multivariate parameters $(\mathcal{N}(\mu_1, \sigma_1^2), \mathcal{N}(\mu_2, \sigma_2^2), ..., \mathcal{N}(\mu_{g-1}, \sigma_{g-1}^2)$. This encoding of trajectory of past decisions plays an important role to help balance the *exploration* vs *exploitation* tradeoff and prevent CMA-ES from getting stuck in local optimas.

$$R_C^g = \underbrace{\begin{bmatrix} r_g^{f_1,1} & r_g^{f_1,2} & \cdots & r_g^{f_1,C} \\ \vdots & \vdots & \ddots & \vdots \\ r_g^{f_n,1} & r_g^{f_n,2} & \cdots & r_g^{f_n,C} \end{bmatrix}}_{\text{Each } \textbf{column} \text{ is a flattened regulated injection rate } (h_n)} \sim \mathcal{N}(\mu_g, \sigma_g^2) \tag{7}$$

$$\mathcal{N}(\mu_{g+1}, \sigma_{g+1}^2) = CMA(\{\mathcal{N}(\mu_1, \sigma_1^2), \mathcal{N}(\mu_2, \sigma_2^2), ..., \mathcal{N}(\mu_g, \sigma_g^2)\}, R_C^g, R_{\lambda:C}^g) \tag{8}$$

Representative updates to a 2D Gaussian distribution for `multivariate` learning of regulation for a 2x2 HopliteBP NoC are shown in Fig 6. As a black box optimization, CMA-ES enables us to learn best R in a large search space of $(0,1]^{N^4}$ without requiring any explicit knowledge of the QoR function or the myriad functions in the Hoplite analysis suite. This foregoes the need for any calculation of gradients and allows for a well composed interaction with other black box learners (like MLE for switch configuration); thus allowing for symbiotic nested learning techniques over multiple NoC parameters.

5.3 Nested learning

In this section, we explain the formulation of nested learning, where multiple parameters of the NoC can be learnt to optimise for a specific application flowset. We focus on learning regulation rates as well as switch level configuration of the NoC *in-tandem*; with each influencing the other's search space and having a direct impact on the learnt parameters. This mutual dependence arises from the direct coupling of regulation and switch configuration on the performance of the NoC. When learning for a specific application flowset, changing one set of parameters (regulation rates for example) will influence potential application routing, thus shaping the search space for the other parameter set (switch configuration for example).

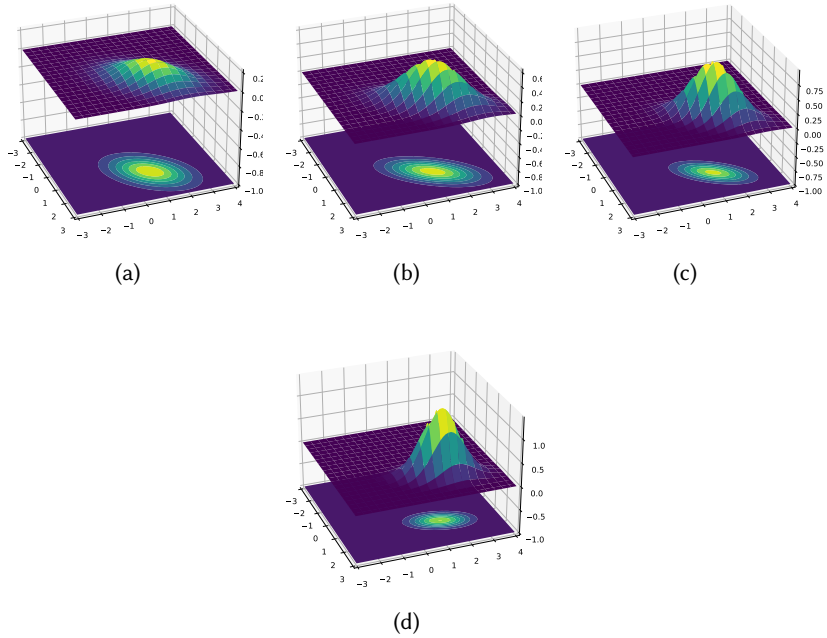


Fig. 6. Evolution of a 2D gaussian distribution when learning regulation rates for a 2x2 HopliteBP NoC with 2 traces of data block size 1 in a RANDOM flowset. Note the convergence of the distribution over epochs as CMA-ES progressively learns better solutions.

Result: R reg., S switch conf.
CMA.init;
while *not CMA.converge* **do**
  CMA.gen_samples;
  for s_{CMA} *in CMA.samples* **do**
    MLE.init;
    while *not MLE.converge* **do**
      MLE.gen_samples;
      for s_{MLE} *in MLE.samples* **do**
        qor=calc(s_{CMA}, s_{MLE});
        MLE.tell(qor);
      MLE.fit();
    qor=calc(s_{CMA}, MLE.best);
    CMA.tell(qor);
  CMA.fit();

Algorithm 1 explains the algorithm for rate learning along with learning switch level configuration for Hoplite NoCs. The objective of this algorithm is to find rate R, switch configurations S such that (R, S)= argmax($Q(H(F, \rho, R, S))$). Note that `multivariate learning` has a much larger search space $\in (0, 1]^{N^4}$ compared to switch learning which has a search space of 2^{N^2}. Also a gaussian distribution (for rates) is computationally more challenging than a Bernoulli distribution (for switches). Thus, we nest the switch configuration learning algorithm (*MLE*) inside rate learning (*CMA*).This decision helps cut down learning times many folds since the much larger search space of rate learning will only need to converge once; with the simpler switch learning converging to generate best switch configuration for each rate candidate generated by CMA. The `fit` functions used by respective algorithms (*CMA* and *MLE*) perform the action of selection of fittest candidates out of *CMA*.samples and *MLE*.samples respectively, followed by evolution of their respective distributions.

Since the nested search space can rapidly explode in size with increasing NoC size for challenging flowsets, we prepend Algorithm 1 with steps to shrink the search space. We do this for 2 reasons: 1. To remove search space with obviously infeasible solutions (for example, little to no regulation for all traces) and 2. Provide a better seed/starting position to `multivariate learning` for a tighter initial spread of gaussian distribution. To this end, we first introduce a universal regulated injection rate of 1 (implying no regulation) and apply it to all traces in the application flowset. We keep *halving* this rate until we achieve feasibility in the NoC. This simple step helps us prune the search space of regulation rates to only include feasible solutions. Since we keep halving the rate until feasibility is achieved, the time complexity is logarithmic in nature; taking only $|\log_2(K)|$ steps to find the first feasible rate K. A simple search, as used in [17], exhibits linear asymptotic time complexity in contrast; taking $(\frac{1-K}{G})$ steps, with G ∈ (0,1) being the resolution of the search increment. We use this first "*feasible*" rate to seed a `univariate` learning of regulation rates. We learn this `univariate` rate until convergence and use this to seed our final phase of `multivariate` learning. Although `univariate` learning offers much less fidelity and does not produce an optimal solution, it seeds `multivariate` learning's starting parameters such that expected coordinate-wise distance to the true optimum is minimised. Without this seeding, the CMA optimizer can exhibit numerical instability; with the underlying probability density function lacking any coherent structure pertinent to the problem and hence generating solutions on pure chance. Asymptotically, the optimizer can take infinite steps before converging to a valid solution. Hence, seeding results in a tighter (implying smaller standard deviation) multivariate gaussian distribution, taking fewer steps for `multivariate` learning to converge. We evaluate this in greater details in section 6.2.3 and Fig. 22.

6 Evaluation

6.1 Methodology

For switch learning, 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 MLE switch learning implementation 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. For regulation rate learning, we implement the learning suite using the python based `cma` library. We parallelize computation of QoR for each sample candidate generated in an epoch by spawning individual `Process` objects using the `multiprocessing` library. Each spawned `Process` can either calculate the QoR directly for vanilla Hoplite configurations (no switch learning; section 6.2.3) or learn switch configurations as part of the internal loop of Algorithm 1 (nested learning; 6.2.2).

We evaluate our entire 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 [20] and `Graph` [16] analytics. The extracted traces exhibit a rich diversity of co-relation and bandwidth requirements; with load factors of 37-92% between the benchmarks. Furthermore, the traces in these real benchmarks are a snapshot of real edge-point data, allowing us to evaluate our designs on ability to route data that closely mimics interactions of end-points in a production setting.

In addition to the static analysis above, we also implement and verify the entire MLE generated 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.

6.2 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 study the effects of:

- Standalone learning of switch level configurations with regulation rate as a user controlled parameter for evaluation (Section 6.2.1).
- In-tandem learning of regulation rates and switch configurations (Section 6.2.2)
- Standalone learning of regulation rates with the switch configuration preemptively set to all HopliteBuf or HopliteBP (Section 6.2.3).

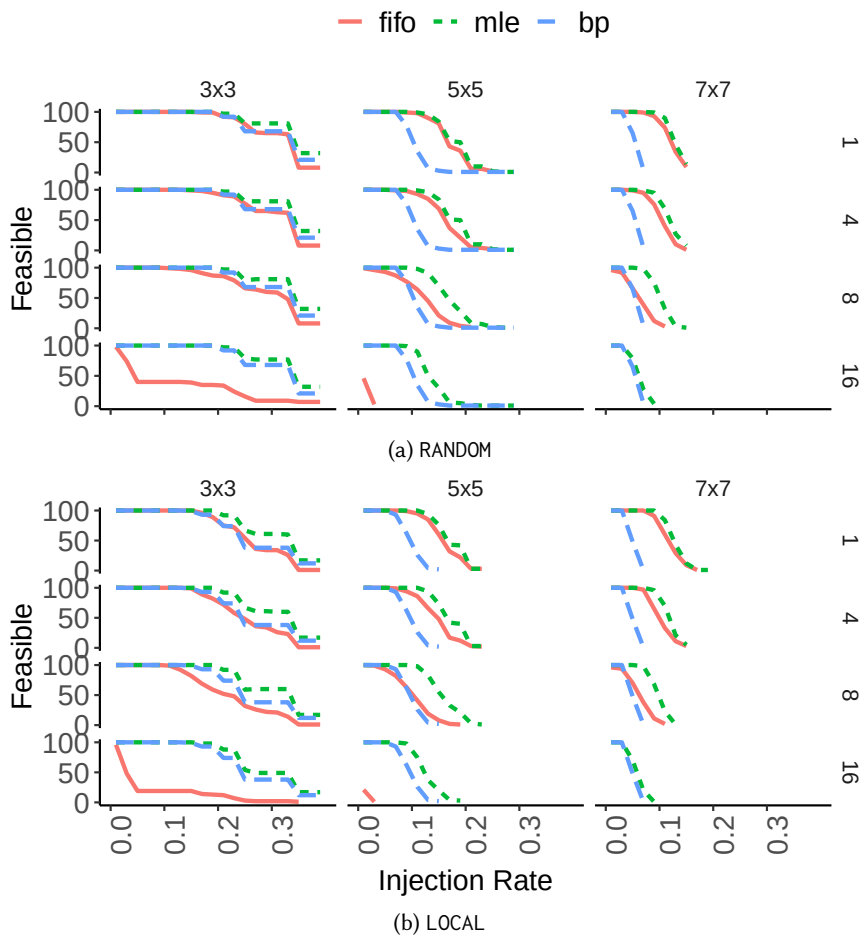
We are primarily interested in determining network feasibility (bounded latency), FIFO size needed, worst-case latency, fitness of the solution, and time required to optimize the NoC.

6.2.1 *Learning Switch level Configuration only*

For the following set of experiments, we focus on only learning switch level configuration to generate hybrid NoCs. We treat the regulated injection rate rate as a user set universal parameter and quantify the resulting NoCs on their ability to optimised for the requested QoR over a linear range (between 0 and 1) of these rates.

6.2.1.1 Learning for Feasibility: In Fig. 7, 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. 7. Feasible set of RANDOM and LOCAL flowsets on system sizes 3×3–7×7 (***across columns***) and data block sizes 1–16 (***across rows***). **Note x and y axes are shared vertically and horizontally respectively**

6.2.1.2 Learning for Worst-Case Latency: Next, in Fig. 8, 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.
- 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.

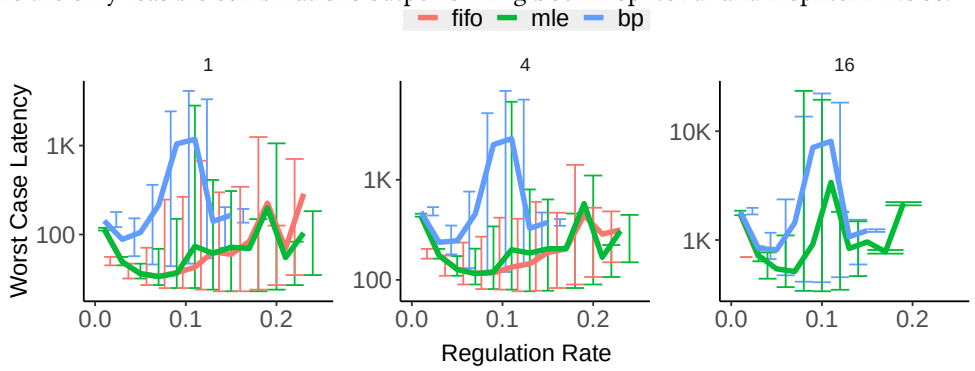


Fig. 8. Worst-case latency as a function of regulation rate for 100 synthetic LOCAL flowsets for various data block sizes.

6.2.1.3 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 Fig. 9, 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 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 Fig. 10 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.

6.2.1.4 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 Fig. 11, 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

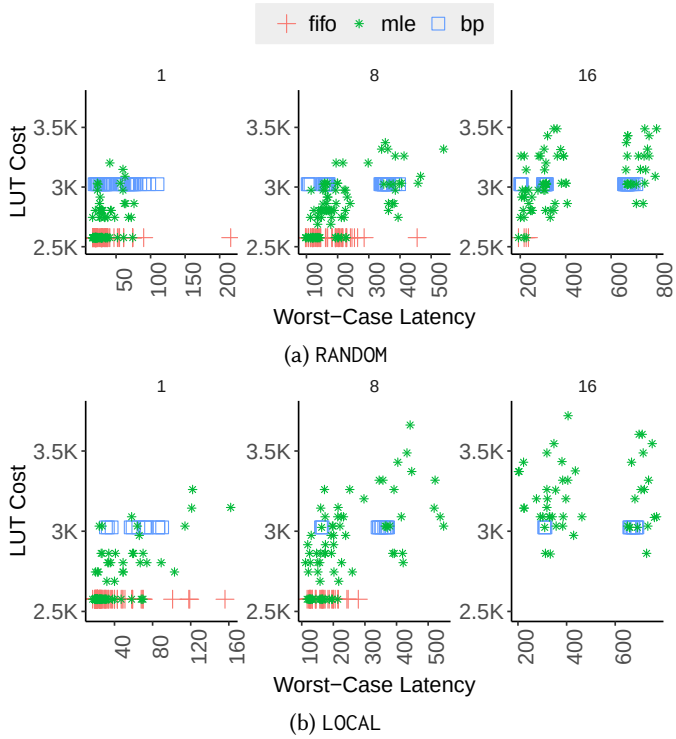


Fig. 9. LUT Cost-Worst Case Latency tradeoffs for a 4×4 NoC with 0.17 regulation rate for 100 synthetic RANDOM and LOCAL flowsets across various block sizes.

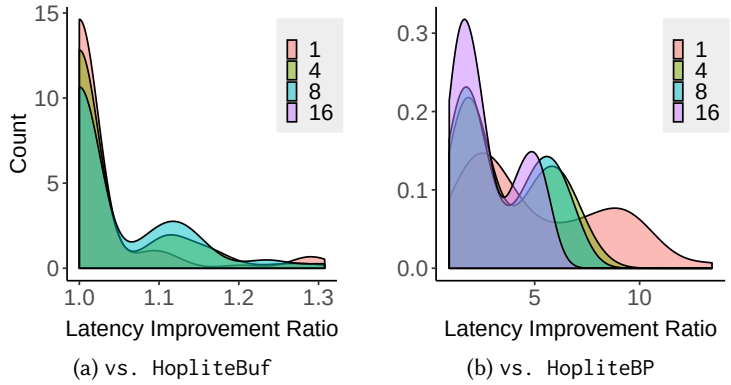


Fig. 10. Cumulative distribution of improvements of worst-case latency MLE NoC over HopliteBuf and HopliteBP

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 2), 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.

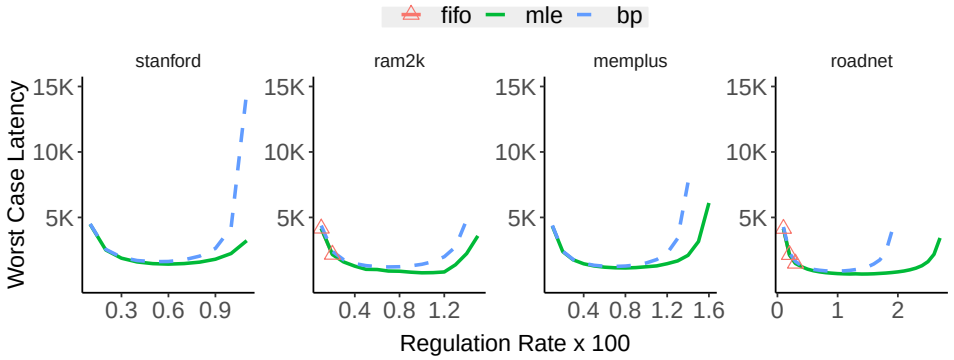


Fig. 11. Latency Scaling of a 4x4 NoC for Graph/ SpMV traces

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

6.2.1.5 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 Fig. 12, 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 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

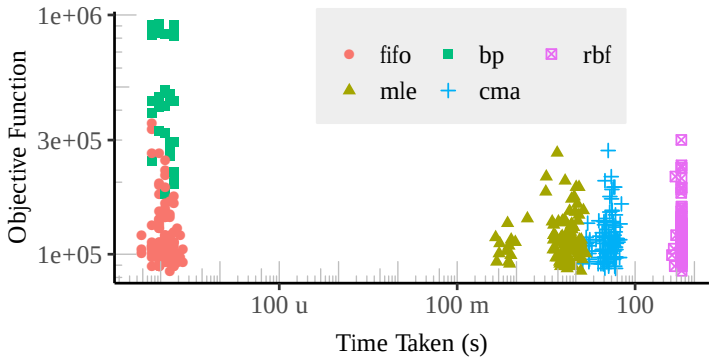
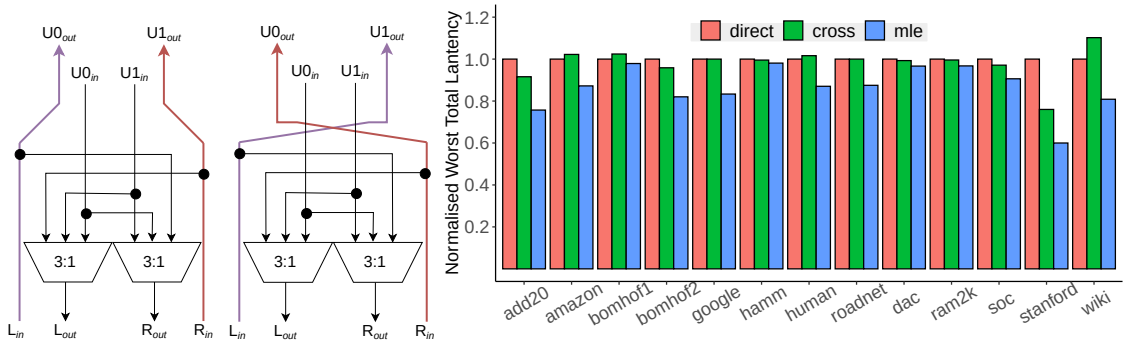


Fig. 12. Solution quality and time taken by MLE optimizer across flowsets mapped to a 5×5 NoC with 0.1 regulation rate

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.

6.2.1.6 Adapting MLE for NoCs with different topologies and parameters: In this section, we take a look at MLE's ability to optimise diverse NoC design parameters for NoCs with markedly different topologies. For this case study, we focus on the Butterfly Fat Tree crossbar (BFT3) topology in [18]. A BFT3 NoC with N endpoints is made up of $log_2(N) * \frac{N}{2}$ unique *pi* switches. We refer the reader to [18] for further information on the BFT3 NoC and *pi* switches. [18] configures a packet climbing a *pi* switch to have a `direct` routing; packets entering a *pi* switch's left port are routed to the left port of U (L->U0) with a similar connection for packets climbing a switch from the right (R->U1), as shown in Fig. 13a. Alternatively, `cross` routing connects L->U1 and R->U0, shown in Fig. 13b. In this work instead, we use MLE to configure the routing of the crossbar, choosing between `cross` or `direct`, on a *per-switch* basis for each of the $log_2(N) * \frac{N}{2}$ switches, tuned to optimise a specified QoR. Contrasting Fig. 13a and 13b, MLE introduces no overheads since only direct connectivity is changed. Furthermore, properties of the original work: live-lock freedom and in-order packet delivery are also preserved. We optimise a BFT3 NoC with 16 endpoints to minimise the total worst latency while routing `real` benchmarks at 100% injection rate in Fig. 13c. We observe that an application optimised BFT3 achieves latencies that are ≈1.1-1.7× lower than the vanilla `direct` and `cross` BFT3 versions, strongly suggesting that the MLE framework of this work can be leveraged for other NoC topologies and tuning different parameter spaces.

6.2.1.7 Comparing an ASIC implementation with Timing Predictable NoCs: In this section, we compare and analyze a potential ASIC implementation of an 8x8 Hoplitebuf+BP NoC where the NoC switches can be fabricated once and be configured to operate in FIFO or backpressure mode, configured on a per-switch basis for the application at hand. We compare the result of MLE optimised analysis of the hoplite NoC and compare it against the state-of-the-art in timing predictable NoCs: mainly SurfNoC [28] and PhaseNoC [23]. We evaluate the NoCs on 2 fronts: 1. The Area overhead for each router (Table 3) and 2. Average routing latency (Fig. 15). For estimating the area of the HopliteBuf+BP switch, we model each 6-LUT of the switch using transistors [24], with a representative design for a 3-LUT connection shown in Fig. 14 (black is `0-through` and white is `1-through` transistor). We estimate area consumption of the resulting design using Intel 45 nm technology with a transistor density of $3.3x10^6$. We observe that the HopliteBuf+BP (MLE-1 in Table 3) switch is 3-3.7× smaller than its counterparts, thus allowing us to replicate the NoC up to 3 times (MLE-3) and tripling the available routing bandwidth, while still maintaining area parity



(a) `Direct` Routing (b) `Cross` Routing (c) MLE configured BFT3 crossbar against `direct` and `cross` vanilla crossbars

Fig. 13. Analyzing MLE's ability in optimising for the parameter design space of a Butterfly Fat Tree crossbar NoC.

with Phase and Surf NoCs. This is especially useful for increasing routing feasibility when routing becomes infeasible with increasing injection rates, owing to the lightweight flow controls of the switches (compared to Surf and Phase).

For latency comparison in Fig. 15, we compare the average latencies of each trace in the application flowset with random-uniform communication patterns with a data block size of 6, where each node in the NoC randomly chooses another destination node in the NoC. We generate 100 such random application flowsets to eliminate any bias in the distributions and average the results over these application flowsets. The data for Phase and Surf NoCs is extracted from [23]. We add additional channels (MLE-2, MLE-3) whenever the MLE generated NoC becomes infeasible with increasing injection rate. Note that even with replicated channels, the MLE optimised HopliteBuf+BP ASIC NoC is still cheaper than the Surf and Phase NoCs (Table 3). We observe in Fig. 15 that MLE generated NoC analysis results in average latencies that are 1.3-12× lower than Phase and Surf NoC variants, owing to the application aware tuning of the NoC and always available routing bandwidth flow control of buffering and backpressure, without any TDM scheduled routing breaks.

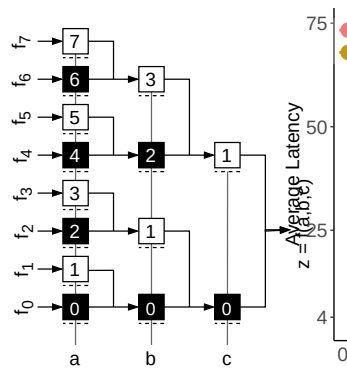


Fig. 14. Design of a 3-LUT [24]

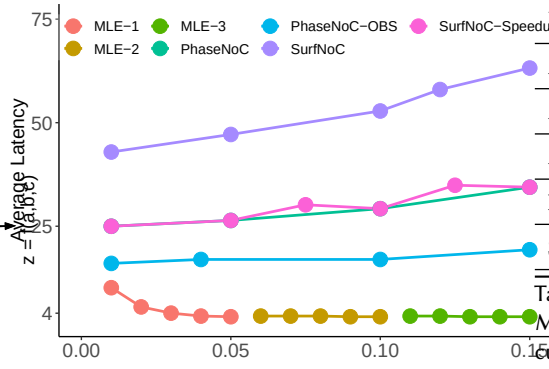


Fig. 15. Average Latency vs Rate. Phase and SurfNoC from [22]

NoC	Area
MLE-1	**0.009**
MLE-2	**0.019**
MLE-3	**0.028**
PhaseNoC [23]	0.027
SurfNoC [28]	0.033

Table 3. Area in mm^2. MLE NoCs' area calculated using Intel 45 nm process ($3.3x10^6$ transistors/mm^2)

6.2.2 *In-Tandem Learning Regulation Rates and Switch Configurations*

In this section, we investigate the effectiveness of learning regulation rates while also learning

switch configuration on a per-switch basis. We compare this *nested* learning against standalone switch learning of section 6.2.1. We contrast `multivariate` rate learning with `univariate` rate learning against a variety of `RANDOM`, `LOCAL` and `REAL` benchmarks. For both learning types, we use MLE as our algorithm of choice to learn switch level configurations. Note that the previous section 6.2.1 separates out the regulation rate as a single value determined by the user. `Univariate` rate learning is an extension of linear searching of the previous section 6.2.1; in that the rate, while still being a single value that regulates all traffic, is now learnt as part of the optimization problem instead of being linearly searched over a range with some fixed increment. If this increment was made infinitesimally small and the rate point with the best QoR chosen, then linear search and `univariate` would generate equivalent solutions.

It is also helpful to recall that since both rates (`univariate` and of section 6.2.1) are a single value that are universally applied to all regulators in the NoC, they do not offer the level of fidelity required to individually regulate NoC bandwidth to each traffic trace. `Multivariate` learning instead learns unique rates for each *src-dst* traffic trace pair. Over a range of benchmarks, we aim to investigate if `multivariate` rate learning can implicitly identify potential congestion hotspots within the NoC and tweak regulation of offending individual traffic to outperform `univariate` learning by achieving lower routing latency while sustaining higher rates across a range of benchmark flowsets.

6.2.2.1 Routing `RANDOM`, `LOCAL` *and* `REAL` *Benchmarks:* We first compare `multivariate` and `univariate` rates by evaluating their performance on numerous `RANDOM`, `LOCAL` and `REAL` traffic benchmarks of varying combinations data block and NoC sizes. We now list a few important differences between `multivariate` and `univariate` learning, along with their impact on performance of the final NoC design in Figs. 16 – 18:

- As NoC sizes increase, `multivariate` is afforded an increased capacity to tune regulation rates due to increase in parameter space N^4. This larger parameter space allows it to work with MLE at a more granular level to optimise for the requested QoR. In contrast, `univariate's` parameter space remains the same. This implies that the effectiveness of `univariate` decreases with NoC size; it is unable to individually respond to dramatically different routing conditions between numerous local regions of a bigger NoC. Thus, `multivariate's` increased tuning capability allows it implicitly tag locally congested regions and throttle down contention within these local NoC zones by surgically regulating only the offending traces.

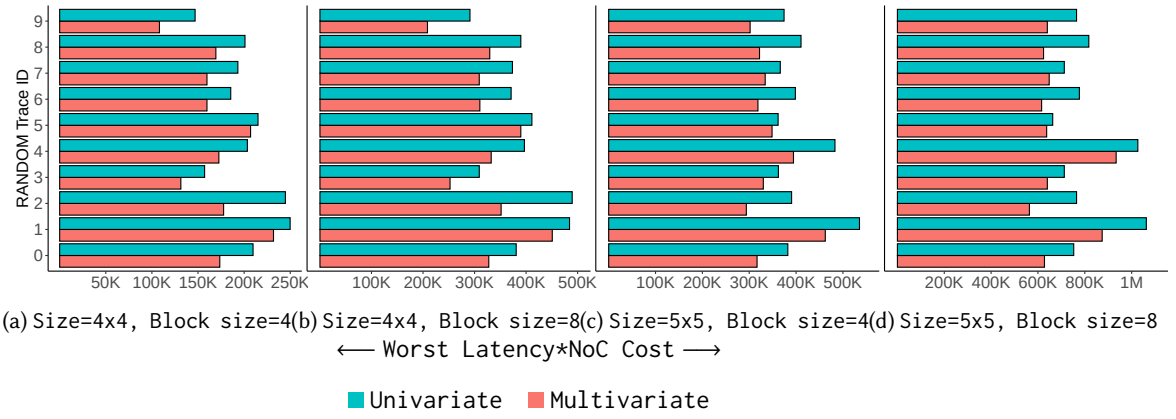


Fig. 16. QoR (`Worst case Latency*NoC Cost`) achieved by univariate and multivariate rate learning strategies, compared for data block sizes of 4, 8 and NoC sizes of 4x4, 5x5 for 10 different `RANDOM` traces.

- Increasing data block sizes can quickly overwhelm the cheaper HopliteBuf (FIFO) switches throughout the NoC. That pressure can be relieved by MLE's switch discovery by selectively replacing these pinched HopliteBuf switches with HopliteBP switches. However, `univariate` unilaterally lower rates for all traces which in turn increases HopliteBuf's feasibility, thus not exposing these switch level decisions to the MLE algorithm. In contrast, `multivariate` and MLE are able to work in tandem to lower rates as a *last resort*; only where pressure cannot be relieved by *Buf->BP* swaps. Thus, the search space of MLE's switch level learning and `multivariate's` rate learning are well composed when they interact; with each being in lock-step with other, thus allowing for a more efficient optimization for the requested QoR.

We now compare the quality of solutions generated by `multivariate` and `univariate` learning for different `RANDOM`, `LOCAL` and `REAL` benchmarks. Rate learning and switch learning algorithms are configured to optimise the combined QoR of $wclatency * cost$:

- In Fig. 16, we plot the efficacy of routing 10 different `RANDOM` traffic benchmarks each across NoC sizes of 4x4 and 5x5 while varying data block sizes between 4 and 8. We observe that `multivariate` rates outperform `univariate` rates by upto ≈1.5×.
- In Fig. 17, we plot the efficacy of routing 10 different `LOCAL` traffic benchmarks each across NoC sizes of 4x4 and 5x5 while varying data block sizes between 4 and 8. We observe that `multivariate` rates outperform `univariate` rates by upto ≈1.3×.
- In Fig. 18, we plot the efficacy of routing 14 different `REAL` traffic benchmarks from `Graph` and `SpMV` suites in progressively difficult routing conditions; increasing NoC sizes to 4x4 and increasing data block sizes from 1 to 4. These `REAL` benchmarks are characterised by very high load factors (% of active traces in benchmark out of N^4 maximum). As noted in section 6.2.1.4, this leads to traffic traces within the NoC to interact in a *spaghetti like* effect, where multiple traffic traces interact with each other at multiple switch points. This results in any contention ultimately affecting every other traffic trace in the NoC. While linearly searching over regulation rates in sec 6.2.1.4, this affect was evident in low feasibility and exploding worst case latencies. Both `univariate` and `multivariate` are able to learn regulation rates to best mitigate this effect, achieving feasible combination of rates and switch configurations across all 14 benchmarks. Continuing the trend observed in `RANDOM` and `LOCAL`,

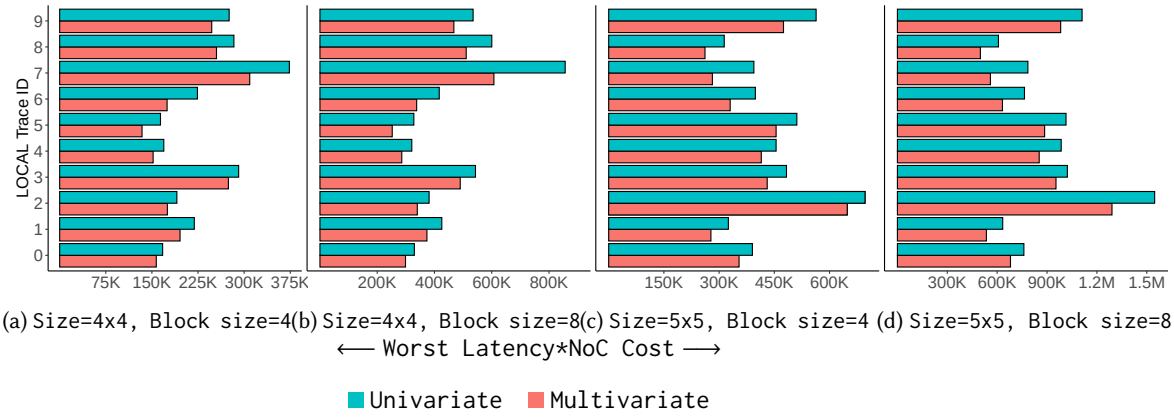


Fig. 17. QoR (`Worst case Latency*NoC Cost`) achieved by univariate and multivariate rate learning strategies, compared for data block sizes of 4, 8 and NoC sizes of 4x4, 5x5 for 10 different `LOCAL` traces.

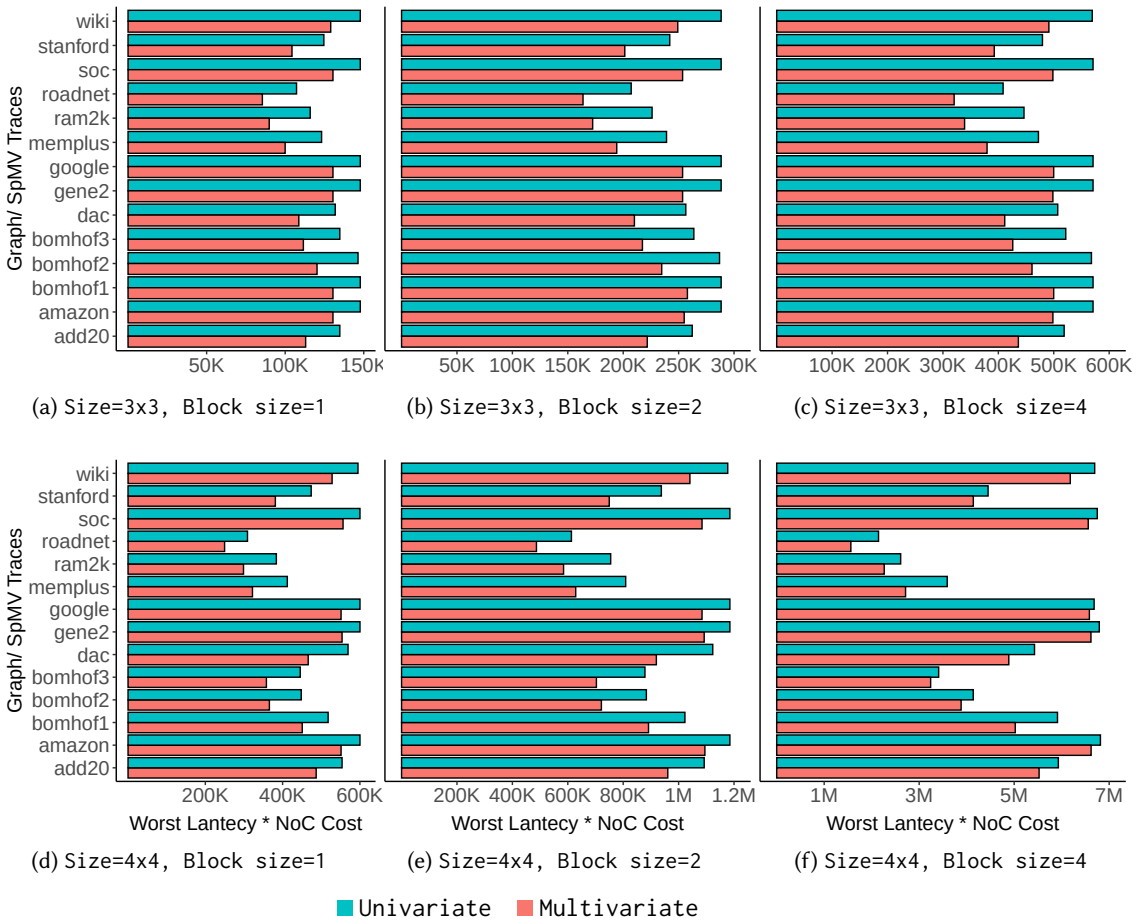


Fig. 18. QoR (Worst case Latency*NoC Cost) achieved by univariate and multivariate rate learning strategies, compared for data block sizes of 1,2 and 4 (horizontal facets) and NoC sizes of 3 and 4 (vertical facets) for REAL traces.

multivariate outperforms univariate across the board by as much as ≈1.2×, owing to its better ability to detangle (by regulating) conflicting traces within the NoC.

6.2.2.2 Analyzing Learnt Rates: With this experiment, we compare and contrast the breakdown of distribution of rates, specifically 1. if multivariate rates are able to able to achieve a wider spread of rates, implying traffic aware regulation and 2. if multivariate is able to achieve, on average, higher rates than univariate. We focus on REAL benchmarks for this experiment due to their particularly high susceptibility to regulation; owing to their high load factor induced *spaghetti like* contention. We also choose real benchmarks for this experiment due to the high diversity of traffic behaviour within a benchmark. This allows us to evaluate the degree of acute homogeneity introduced by univariate learning and its worsening impact by achieving much lower rates and contrasting this with multivariate's ability to instead exploit this diversity for faster rates overall. For this, we plot, for a NoC of size 4x4, the range of learnt multivariate rates for all 14 REAL benchmarks while increasing the data block sizes from 1, 2 and finally to 4 in Fig. 19. We also plot the average multivariate rates and compare it with univariate.

We observe that the mean `multivariate` rate is higher than the univariate rate for nearly every benchmark across every data block size; up to ≈1.4× higher. This suggests that `multivariate` learns to first lower the rates of most contentious traces in the benchmarks, thus allowing other traces to achieve ≈3.1× higher regulated injection rates, further evidence that turning down the "heat" in certain parts of the network helps other traces immensely. This is also evidenced, in benchmarks like `dac`, `roadnet` etc, by the observation that the *minimum* of the `multivariate` rates is actually ≈1.8× lower than its `univariate` counterpart, despite its average being higher than `univariate`. Another implication of this is that the "one size fits" all strategy of having each trace be regulated by the same amount leaves quite a bit of NoC bandwidth on the table, explained by the NoC being pushed more easily into infeasibility because any "superhot" contention zone within the NoC affects the entire interconnect, thus forming a bottleneck for regulation. `Multivariate`, on the other hand, will simply lower regulation rate for traces involved in such contentions.

6.2.3 *Learning Regulation Rates for Vanilla Hoplite NoCs*

In this section, we analyze the effectiveness of `multivariate` over `univariate` learning when learning regulation rates for traces that need to be routed on a vanilla Hoplite configuration, that is, all switches are either HopliteBuf (fifo) or HopliteBP (bp). Vanilla HopliteBuf NoCs' FIFOs can quickly fill up as a result of increased contention induced buffering. HopliteBP NoCs suffer from pessimistic analysis, born out of long chain of backpressure signals that can throttle all links across a row. We investigate if `multivariate` and `univariate` learning can help alleviate some of these inherent weaknesses of vanilla NoCs, while also comparing both rate learning techniques against each other.

6.2.3.1 Analyzing Distribution of Latency Improvement: In Fig. 20, we plot the latency improvement distribution of `multivariate` over `univariate` while routing 100 different `RANDOM` benchmarks (Fig. 20a), 100 different `LOCAL` benchmarks (Fig. 20b) and all 14 `REAL` benchmarks (Fig. 20c) on 4x4 HopliteBuf (fifo) and HopliteBP (bp) NoCs while varying data block sizes between 1, 2 and 4. We observe that `multivariate` achieves better latency by as much as ≈1.6×. As data block size increases, we see bigger latency improvements (the distribution shifts to the right) across all 3 benchmarks and for both Hoplite designs. Recall that in the case of HopliteBuf, FIFOs are able to absorb packets over a larger time horizon for a low data block size of 1. Hence, in this case, we see `univariate` being quite competitive across all benchmarks, with improvement limited to

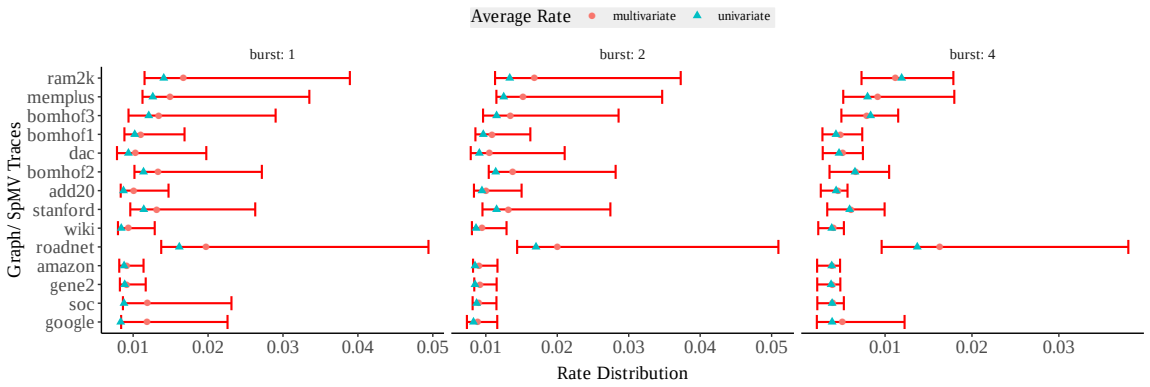
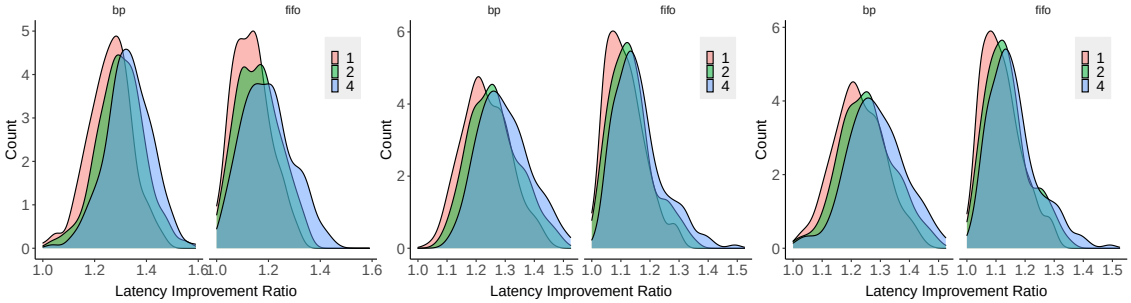


Fig. 19. Range (as whisker plot) and average rates of learnt `multivariate` rates on a *per trace basis* along with `univariate` rates for `REAL` benchmarks for a NoC of size 4x4, over a range of data block sizes (`burst`).

≈1.3×. But as data block size increases, FIFO's quickly run out of capacity to absorb packets, with `multivariate` continuing to improve over `univariate`. While not being as sensitive to data block size owing to lack of FIFOs in their switch design, HopliteBP also has `Multivariate` outperform `univariate` by ≈1.5×, helped by `multivariate's` more granular regulation resulting in fewer row spanning backpressure stalls.



(a) Improvements for 100 RANDOM traces (b) Improvements for 100 LOCAL traces (c) Improvements for REAL traces

Fig. 20. Latency improvements of `multivariate` learning over `univariate` learning, for vanilla HopliteBuf (`fifo`) and HopliteBP (bp) NoCs of size 4x4, routing RANDOM, LOCAL and REAL traces.

6.2.3.2 Routing REAL *Benchmarks:* In this section, we dive deeper into routing of REAL benchmarks by vanilla HopliteBuf (fifo) and HopliteBP (bp), with regulation rates being learnt by `univariate` and `multivariate` learning. In Fig. 21, we plot the worst latency for routing 14 different REAL benchmarks from `Graph/ SpMV` suites on vanilla HopliteBuf and HopliteBP NoCs on size 4x4, while varying data block sizes between 1,2 and 4.

We observe that latency progressively increases for HopliteBP with increasing data block size, with `multivariate` tapping down latency by as much as ≈1.2× compared to `univariate`. We observe a similar scaling behaviour in the case of HopliteBuf with data block size of 1 and 2, with `multivariate` resulting in ≈1.3× lower latency. However, we observe that both `multivariate` and `univariate` perform very poorly when the data block size is increased to 4 for HopliteBuf; with only `ram2k` and `roadnet` resulting in feasible designs. All other benchmarks are not even feasible on a vanilla HopliteBuf, despite best efforts by regulation learning algorithms. We have previously discussed this artifact of FIFOs filling up quickly in the face of contention and increasing data block size in sec. 3. This affect is particularly amplified in the case of REAL benchmarks owing to their high load factor. This highlights the importance of combining regulation rate learning with switch level discovery. While `multivariate` is able to better manage contention at a traffic trace level by tuning regulation of offending traces involved in the contention, it is not as effective at mitigating contention induced in the NoC as an artifact of the underlying switch architecture. Thus learning switch configurations in tandem with regulation rates works best; as similar experiments of Fig. 18 were able to discover feasible NoC designs and regulation rates for all 14 benchmarks in the REAL suite.

6.2.3.3 Understanding `Multivariate` *Rate Learning:* In Fig. 22, we present how `multivariate` rates are learn using Covariance Matrix Adaptation for a 4x4 HopliteBP NoC routing `roadnet` benchmark with a data block size of 1. As discussed in section 5.3, `multivariate` learning is split into three phases: 1. Rate Halving, 2. Univariate rate learning and finally 3. Multivariate rate learning.

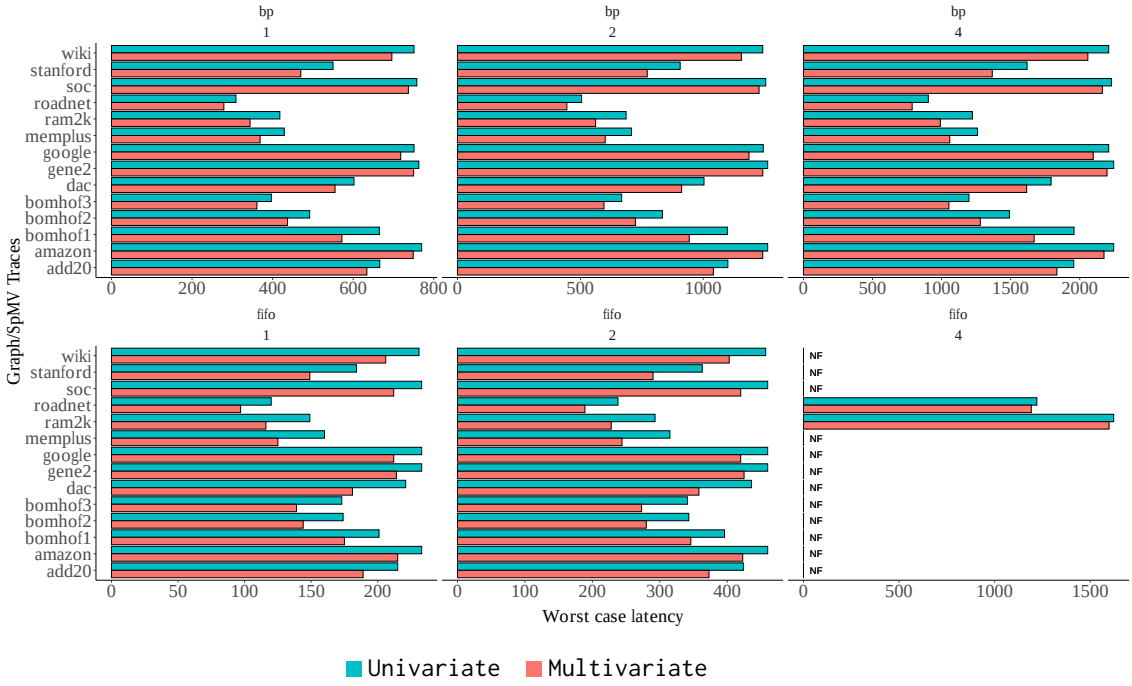
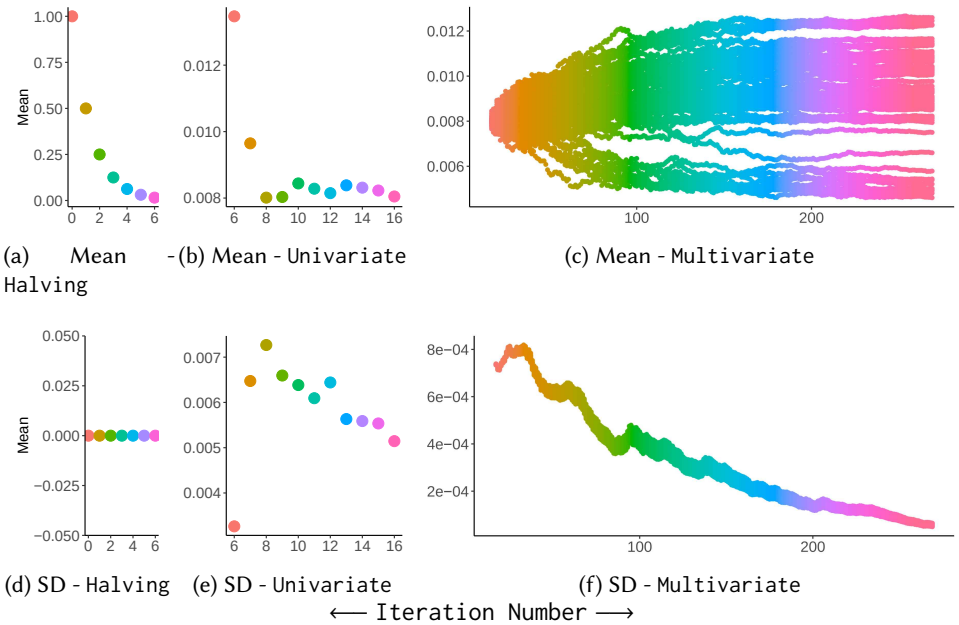


Fig. 21. `Univariate` and `multivariate` rate learning compared for vanilla HopliteBuf (fifo) and (HopliteBP) NoCs of size 4x4 while routing REAL traces of data block size of 1,2 and 4. "NF" tagged flowsets could not be routed feasibly for both learning types on vanilla HopliteBuf NoCs.

The phases of Rate Halving and Univariate learning are motivated by reducing the search space for the `multivariate` phase by first finding a universal regulation rate (applied to regulate all traffic traces) that can achieve the best QoR. Rate halving comprises of starting from a universal rate of 1 (no regulation) and halving it every next iteration until the NoC can sustainably route the benchmark at hand. Fig. 22a shows the rate being halved at every iteration until sustainability can be confirmed for all traces in the benchmark. For `roadnet`, this rate is $\frac{1}{2^6}$.At this stage, the rate is modelled as a univariate gaussian distribution with a mean μ (Fig. 22b) and standard deviation σ (Fig. 22e) which is then learnt over multiple iterations to optimise for the requested QoR. The termination condition of the univariate phase is determined by the trajectory of best QoR over past iterations and the convergence of standard deviation (notice it reduce after every iteration to rest at ≈0.005).

Finally, a multivariate gaussian distribution is modelled to represent the rate of each trace individually, starting off with the same mean for very dimension, set equal to the best rate found at the termination of the univariate phase, as shown in Fig. 22c. The standard deviation, in Fig. 22f, for every dimension is initialised to the 10% of `multivariate`'s initial rate . This gaussian distribution's parameters: mean and standard deviation for each dimension, are then learnt over the course of multiple iterations. Notice that the mean rates for every trace diverge to settle at values that best optimise the QoR, spread over the entire range, with some of the final rates being almost ≈3× faster than others. `Multivariate` learning is terminated by analyzing the trajectory of best QoR over the iteration history as well as convergence of standard deviation of each rate, with standard deviations converging from $\approx 7 * 10^{-4}$ to $\approx 5 * 10^{-5}$.



(a) Mean - Halving (b) Mean - Univariate (c) Mean - Multivariate

(d) SD - Halving (e) SD - Univariate (f) SD - Multivariate

Fig. 22. Chronological updates to mean and standard deviation (SD) of a `multivariate` gaussian distribution for a 4x4 HopliteBP NoC routing the `roadnet` benchmark with a data block size of 1. Note the 3 stages of learning: 1. Rate Halving, 2. Univariate and finally the dominant 3. `multivariate`. Also note the 0 SD for rate halving.

7 Conclusions

In this paper, we show how to evolve hybrid FPGA NoC parameters (switch configurations and trace regulation) to deliver a combination of feasibility, worst-case latency, and cost improvements over homogeneous FPGA NoCs. We demonstrate switch learning by combining HopliteBuf, 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. We model rate learning by casting each traffic trace's regulation rate as a random variable from a multivariate gaussian distribution. We using Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) to learn rates in-tandem with switch configuration and achieve achieve ≈1.5× lower cost constrained latency, ≈3.1× faster individual rates and ≈1.4× faster mean rates.

References

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

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

[3] Alberto Costa and Giacomo Nannicini. 2018. RBFOpt: an open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation* 10, 4 (01 Dec 2018), 597–629. https://doi.org/10.1007/s12532-018-0144-7

[4] Tushar Garg, Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2019. HopliteBuf: FPGA NoCs with Provably Stall-Free FIFOs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. ACM, New York, NY, USA, 222–231. https://doi.org/10.1145/3289602.3293917

[5] Tushar Garg, Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2020. HopliteBuf: Network Calculus-Based Design of FPGA NoCs with Provably Stall-Free FIFOs. *ACM Trans. Reconfigurable Technol. Syst.* 13, 2, Article 6 (Feb. 2020), 35 pages. https://doi.org/10.1145/3375899

[6] Nikolaus Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

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

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

[9] N. Kapre and J. Gray. 2015. Hoplite: Building austere overlay NoCs for FPGAs. In *Field Programmable Logic and Applications*. 1–8. https://doi.org/10.1109/FPL.2015.7293956

[10] Nachiket Kapre and Jan Gray. 2017. Hoplite: A Deflection-Routed Directional Torus NoC for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 10, 2, Article 14 (March 2017), 24 pages. https://doi.org/10.1145/3027486

[11] Nachiket Kapre, Harnhua Ng, Kirvy Teo, and Jaco Naude. 2015. 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* (Monterey, California, USA) *(FPGA '15)*. ACM, New York, NY, USA, 23–26. https://doi.org/10.1145/2684746.2689081

[12] M. Karol, M. Hluchyj, and S. Morgan. 1987. Input Versus Output Queueing on a Space-Division Packet Switch. *IEEE Transactions on Communications* 35, 12 (1987), 1347–1356.

[13] Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christoph Müller, Kees Goossens, and Jens Sparsø. 2015. Argo: A real-time network-on-chip architecture with an efficient GALS implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 2 (2015), 479–492.

[14] Gwangsun Kim, Michael Mihn-Jong Lee, John Kim, Jae W Lee, Dennis Abts, and Michael Marty. 2012. Low-overhead network-on-chip support for location-oblivious task placement. *IEEE Trans. Comput.* 63, 6 (2012), 1487–1500.

[15] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag.

[16] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection.

[17] Gurshaant Malik, Ian Elmor Lang, Rodolfo Pellizoni, and Nachiket Kapre. 2020. Learn the Switches: Evolving FPGA NoCs with Stall-Free and Backpressure Based Routers. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 18–25.

[18] G. S. Malik and N. Kapre. 2019. 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)*. 154–162.

[19] Michael K Papamichael and James C Hoe. 2012. 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*. ACM, 37–46.

[20] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 27–40.

[21] Tomas Picornell, José Flich, Carles Hernández, and Jose Duato. 2020. Enforcing predictability of many-cores with DCFNoC. *IEEE Trans. Comput.* 70, 2 (2020), 270–283.

[22] Anastasios Psarras, Junghee Lee, Ioannis Seitanidis, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. 2015. PhaseNoC: Versatile network traffic isolation through TDM-scheduled virtual channels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 5 (2015), 844–857.

[23] Anastasios Psarras, I Seitanidis, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. 2015. PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1090–1095.

[24] Ali Sheikholeslami, Ryuji Yoshimura, and P Glenn Gulak. 1998. Look-up tables (luts) for multiple-valued, combinational logic. In *Proceedings. 1998 28th IEEE International Symposium on Multiple-Valued Logic (Cat. No. 98CB36138)*. IEEE, 264–269.

[25] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Brian Gaide, and Ygal Arbel. 2019. Network-on-Chip Programmable Platform in VersalTM ACAP Architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. ACM, New York, NY, USA, 212–221. https://doi.org/10.1145/3289602.3293908

[26] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2017. HopliteRT: An efficient FPGA NoC for real-time applications. In *F. Program. Technol. (ICFPT), 2017 Int. Conf.* IEEE, 64–71.

[27] Wasly, Saud, Pellizzoni, Rodolfo, and Kapre, Nachiket. 2017. Worst Case Latency Analysis for Hoplite FPGA-based NoC. http://hdl.handle.net/10012/12600

[28] Hassan MG Wassel, Ying Gao, Jason K Oberg, Ted Huffmire, Ryan Kastner, Frederic T Chong, and Timothy Sherwood. 2013. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 583–594.