Marathon: Statically-Scheduled Conflict-Free Routing on FPGA Overlay NoCs

Nachiket Kapre
Nanyang Technological University
50 Nanyang Avenue, Singapore 639798
Email: nachiket@ieee.org

***Abstract*— We can improve the performance of deflection-routed FPGA overlay networks-on-chip (NoCs) like Hoplite by as much as 10× (random traffic) at the expense of modest extra storage cost when combining static scheduling with packet switching in an efficient, hybrid manner. Deflection routed bufferless NoCs such as Hoplite, allow extremely lightweight packet switched routers on FPGAs, but suffer from high packet latencies due to deflections under congestion. When the communication workload is known in advance, time-multiplexed routing can offer a faster alternative by eliminating deflections but require expensive storage of routing decisions in context buffers in LUT RAMs. In this paper, we propose a hybrid Marathon NoC that combines the low packet latencies of deflection-free time-multiplexed routing with the low implementation cost of context-free packet-switched Hoplite NoC. The Marathon NoC requires a deterministic routing function to be implemented in the switch along with time-stamped packet injection in the PEs to ensure deflection-free routing in the network. The network also needs a one-time offline static scheduling stage that determines the appropriate time to inject a packet to guarantee conflict-free deflection-free route on the shared network. For random traffic patterns, Marathon outperforms Hoplite by as much as 10× and time multiplexing by as much as 1.2× when considering total communication time at identical area costs. For other synthetic patterns, Marathon outperforms Hoplite in all cases except local pattern and is within 2–5× of best time multiplexing performance at large system sizes. For communication workloads extracted from real-world sparse matrix-vector multiplication kernels, Marathon outperforms both Hoplite and Time Multiplexing by 1.3–2.8×.**

I. Introduction

The management of communication requirements in digital designs is a first-class design concern in the effective use of spatial computing fabrics such as FPGAs. Modern FPGAs can easily support million-LUT designs, and at these scales it is imperative that we find resource-efficient ways to support on-chip communication. This can be in the form of system-level communication to the logic from high-speed external interfaces to memories, PCIe, or network ports (GB/s of I/O bandwidth) as well as application-level intra-FPGA communication (TB/s of bisection bandwidths). NoCs (network-on-chip) are a popular mechanism to deliver shared network resources to communicating spatial applications while transferring packets across the chip in a fast, pipelined manner.

Hoplite [7] is a lean 60–90 LUT, 100 FF, 2.9 ns 32b packet-switched router that paves the way for efficient realization of communication requirements in FPGA applications by careful engineering of the logic architecture to match the capabilities

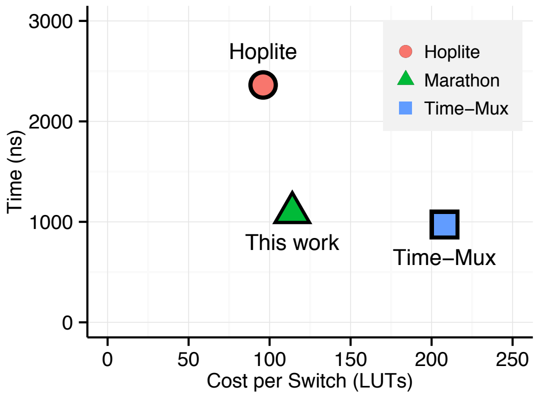


Fig. 1: Evaluating NoC performance-cost trends for an SpMV `add20` workload with 142K packets. Marathon NoC is 2× smaller than Time-Multiplexed NoCs, and 2.5× faster than Hoplite. See Figure 11 for details.

of the FPGA fabric. Hoplite is able to deliver low cost by eliminating expensive packet buffers, simplifying crossbar complexity by choosing a unidirectional torus topology and easing flow control overheads with deflection routing. However, the penalty of deflections [3] under realistic workloads can often result in high worst-case packet latencies. Thus, Hoplite achieves economy of implementation by sacrificing worst-case packet latency that limits its effectiveness for realistic latency-critical applications (10–30% slower than conventional mesh-based NoCs for injection rates above 0.1, severe drop in performance for tornado traffic pattern). Conventional packet-switched NoCs such as CMU Connect [11] and Penn Split-Merge [5] routers are able to avoid these worst-case latency penalties but they are substantially larger (25× larger, 1.7K LUTs) and slower (1.5× slower, 5–10 ns) than Hoplite.

Under these circumstances, we look for inspiration towards time-multiplexed NoCs [8], [15], [13], [9]. Time-Multiplexed NoCs require static knowledge of the communication workload and can offer fast packet delivery without any deflection or buffer wait penalty as each switch is programmed with routing decisions upfront[1]. This is achieved with a static scheduler

[1] *A macro-economist would compare time-multiplexed NoCs to socialist, centrally-managed systems while a packet-switched NoC to a chaotic, capitalist framework.*

that arranges the network traffic to provide each packet of communication a dedicated path through the network. Instead of provisioning the full path for all timeslots, the scheduler generates assigns the exact timeslot in each switch being traversed by the packet along the spatial path to allow efficient reuse of NoC resources. To store the routing decisions, time-multiplexed NoCs require extra storage of routing tables (context memory) in each switch and Processing Element (PE). While we expect spatial FPGA application to generally offer static communication workloads (*i.e.* communication profile of video-processing applications [9], accelerated processing of knowledge-bases [8]), the mapping of storage requirements and programming infrastructure of context memories can become a challenge. LUT-RAMs can work for very small contexts, but we will be forced to steal valuable embedded BlockRAMs away from the FPGA application for larger, real-world communication requirements.

To overcome the twin challenges of high packet latencies (packet switching) and high storage costs (time multiplexing), we propose the hybrid Marathon NoC in this paper. Marathon retains the switching hardware of Hoplite to deliver fast and lean routers. It introduces a minimal scheduling context memory at the PE injection port to record when to inject a packet into the network. Marathon also needs an offline static scheduler that *simulates* the Dimension-Ordered Routing (DOR) routing function of Hoplite routers when finding paths in the network and adjusts the injection timeslot to ensure conflict-free routing. In contrast with time multiplexing, Marathon is unable to choose arbitrary paths in the network and must accept a slight degradation in routing quality. However, as we will discover, our solution still outperforms both Hoplite and classic Time Multiplexing while lowering resource costs by a large margin. For instance, in Figure 1, we show a preview of the area (LUTs) and time (ns) required to route the 142K packet `add20` communication workload (sparse matrix-vector multiplication kernel). Here, Marathon is roughly 2.5$\times$ faster than Hoplite and 30% cheaper than conventional Time Multiplexing.

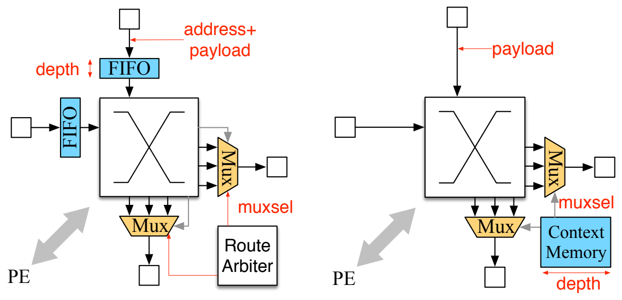
In this paper, we develop Marathon to make the following key contributions:

- Design and engineering of the Marathon Processing Element (PE) RTL[2] and associated implementation on a Virtex-6 LX240T FPGA.
- Development of a space-time router for static scheduling of user-supplied communication workloads that fit bulk-synchronous and dataflow models of communication.
- Quantification of performance and mapping cost of Marathon NoC against Hoplite and Time-Multiplexed FPGA NoCs.

II. Background

In this section, we introduce the key principles behind the operation of FPGA NoCs and discuss our initial design to establish a starting point for further investigation.

[2]The NoC Router RTL is already available from the software repository at *fpga.org/hoplite*.



(a) Packet Switching (b) Time Multiplexing

Fig. 2: Packet-Switched and Time-Multiplexed NoC Routers.

A. Switching Styles

Packet-switched routers process packets arriving from different incoming directions and dispatch them along appropriate outgoing directions based on a routing function and some congestion management strategy. Simple deterministic routing functions such as Dimension Ordered Routing are designed to avoid deadlock in the network by eliminating certain turns in the NoC. The congestion management logic improves routing performance by fairly distributing outgoing port bandwidth using locally available congestion information at the switch. The specific implementations may be buffered [11], [5] or deflection-routed [7]. In Figure 2a, for the buffered packet-switched router, arriving packets must first queue inside the FIFO awaiting their turn. The routing and congestion logic selects the appropriate input-output combination to be serviced in a given clock. For the deflection-routed, packet-switched router (*e.g.* Hoplite), arriving packets do not queue in a FIFO and are instead *hot potato* [4] routed to available output ports. On one hand, this eliminates the need for expensive FPGA buffers, but on the other hand, packets can often deflect multiple times resulting in long worst-case routing delays.

Time-Multiplexed routers also essentially forward incoming packets to outgoing ports but without the need for FIFO storage or any dynamic decision-making logic in the switch. This is possible as the static router is programmed with switching decisions that are made by a traffic compiler offline before the workload is processed by the NoC. This allows the statically-scheduled network to avoid queueing or deflections in the network by keeping a complete view of congestion and occupancy information. In Figure 2b, we observe that cycle-by-cycle routing decisions must be stored in the context memory that supplies control information to the output multiplexer. Time multiplexed routers also do not need any address information to accompany the packet as they already know where the packet is headed thereby saving precious wiring resources. However, the physical implementation size of a time-multiplexed switch varies with the number of cycles required to route a given workload on a given system size. Each switch must provision sufficient storage, best implemented as a compact Xilinx SRL shift register, to store multiplexer decision per cycle.

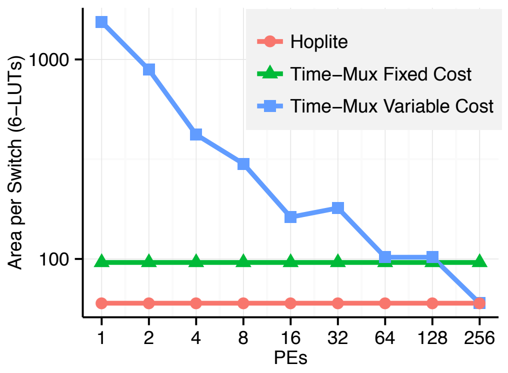


Fig. 3: Area Comparison of Hoplite and Time Multiplexing. 16K messages, `RANDOM` traffic.

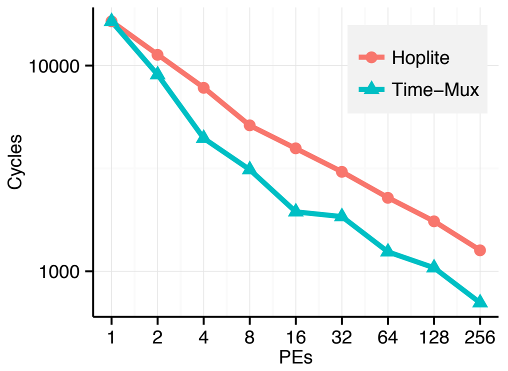


Fig. 4: Performance Limits of Packet Switching when routing 16K packets generated using `RANDOM` pattern.

B. Limitations of Packet-Switched and Time Multiplexed NoCs

In Figure 3, we quantify the resource requirements for a packet-switched Hoplite switch against a time-multiplexed switch. The time-multiplexed switch has a fixed cost component that is purely the cost of implementing wide multiplexers on the FPGA, and a variable component that is the cost of storing the multiplexing decisions in memories (Xilinx SRL-based RAMs). The fixed cost portion is marginally larger than Hoplite as the multiplexers support more routing connections (full crossbar) than dimension-ordered routing (DOR) supported by Hoplite. The bulk of the cost at smaller PE counts comes from context memory required to store the multiplexing decisions. As we need more cycles to route a fixed set of messages with fewer PEs, the size of the context memories is large. As we scale system size to more PEs, the variable cost reduces as we are able to route the complete workload in fewer cycles. However, this only helps at large system sizes.

Next, we identify the limitation of packet switching networks when considering performance in Figure 4. While Hoplite switches are small, the communication workloads suffer a large number of deflections for congested packets. In contrast, time-multiplexed routes are able to avoid congestion by choosing a different timeslot for injection. Packet Switching loses by as much as 2–3× for large PE counts.

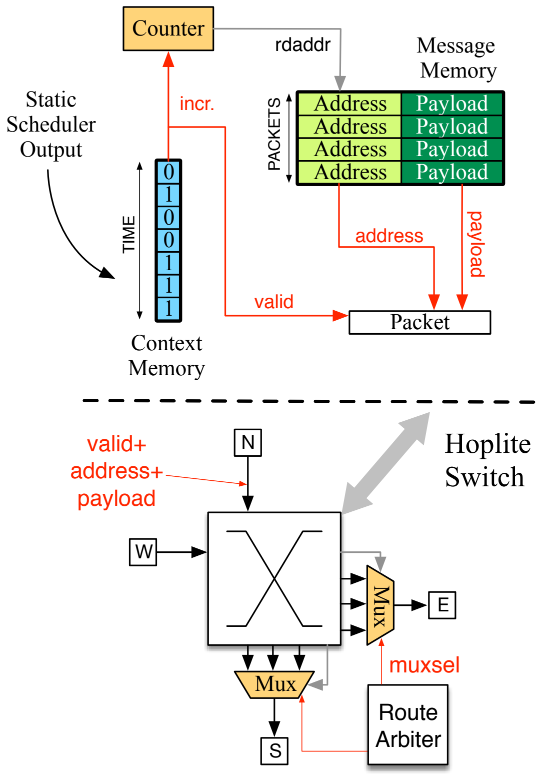


Fig. 5: Hardware Organization of Marathon NoC (Switch and packet injection portion of PE shown, packet receipts in PE are non-blocking).

III. Marathon Soft NoC Architecture

In this section, we introduce our hybrid Marathon architecture and identify the limitations and advantages through an illustrative example. We also describe the software scheduling engine used for time-multiplexed mapping.

A. Hardware

The key idea behind Marathon is to combine the benefits of Hoplite (small resource use) and Time Multiplexing (high-quality schedule). In Figure 5, we see this is achieved by combining the Hoplite switch with a modified PE that injects packets at pre-determined timeslots. The use of a Hoplite switch without FIFO buffers, or context memories implies that NoC implementation costs stay as cheap as Hoplite. It is important that the NoC implements a DOR routing function (or a similar, deterministic function) that is also implemented identically in the software scheduler (details on software scheduling in Section III-C). We also show a portion of the PE which corresponds to the logic required to order packet injection into the NoC. The extra cost imposed by the context memory in the PE is identical to the cost of a time-multiplexed PE and a small percentage of overall time-multiplexed context cost that would otherwise be required. In a statically-scheduled scenario, the entire communication workload needs to known in advance – this corresponds to storing the `Address` field in the `Message Memory` of the PE at configuration time. The `Payload` can be calculated dynamically as required by

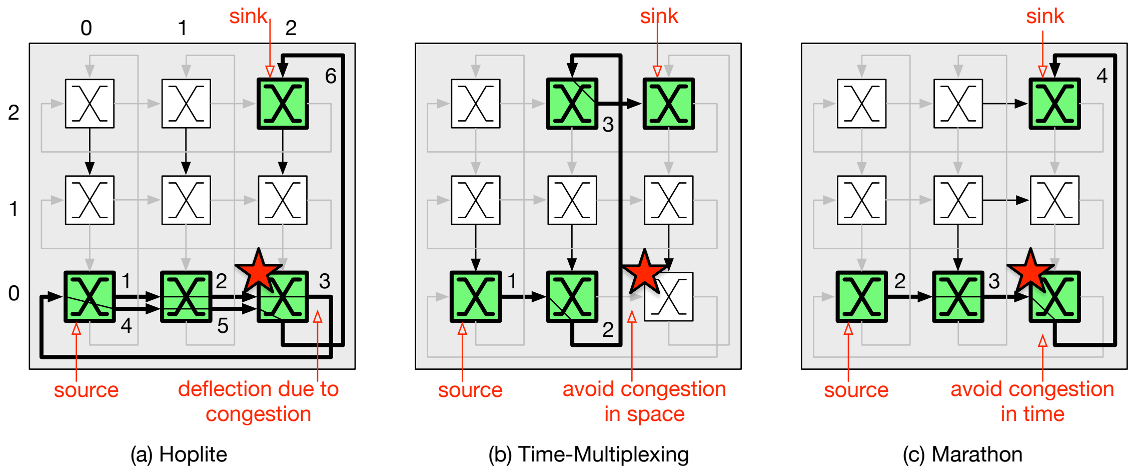


Fig. 6: Comparing paths taken by a packet routing from PE (0,0) to PE (2,2) with congestion at cycle 3 for wrap-around path from PE (0,2) to PE (2,2). Packet-switched route on Hoplite must deflect at cycle 3 and wrap-round in row 0. Time-Multiplexed routes can take a non-Manhattan path that snakes to the destination. Marathon NoC simply delays injection by a cycle while still taking a DOR path without any deflections or conflicts enroute.

the parallel application or workload. As packet injection is scheduled, we need a 1b-wide `Context Memory` to store valid bits for the packet and corresponding increment count indication for the message address generation. The size of this memory will need to be as long the number of cycles required to route the entire communication workload and will generally be larger than the total number of messages to be sent. As the communication structure is known in advance, our static scheduler organized the `Message Memory` in injection order and simply uses the counter to decide which message is to be read for transmission. The `Context Memory` bit serves a dual purpose of incrementing the counter (read address) as well as a valid bit for packet injection. Since contention is impossible, we know that the injected packet will never suffer a deflection and will smoothly arrive at the destination PE using the DOR route. The software scheduler routes the communication graph offline and keep track of used resources when determining packet injection time.

B. *Limits of Marathon*

While it may seem that the Marathon NoC simply gives us the best of both NoC styles, there is a possibility of quality degradation when compared to classic Time Multiplexing. To understand how this tradeoff is encountered, let us consider the example shown in Figure 6. We show a comparative example where we need to send a packet from PE (0,0) in the lower left corner of the NoC to PE (2,2) in the upper right corner. In a conflict-free routing case, we will simply follow the DOR path of moving along the x-dimension first and then turning south in the y-dimension to reach the destination. We assume a hypothetical scenario where the turn from x-to-y dimension encounters congestion *i.e.* some other packet is using the south port at Switch (0,2). In this scenario, the different NoCs will handle congestion in different ways, thereby clarifying their mode of operation:

- **Hoplite**: The packet will simply *deflect* and continue in the x-dimension for another round before returning to the same turn. In this example, after one round-trip in the row, the packet will not see any congestion and be able to take the turn and reach the destination.
- **Time Multiplexing**: The static scheduler will be aware of an impending congestion choke point and the breadth-first search heuristic will direct the scheduler to use an alternate path that avoids the congestion entirely. In this case, we have routed around the congestion in *space* by taking a non-DOR route. Notice how we take two turns for this packet. To enable this route, the Time-Multiplexed scheduler relies on configuring the switch to store the routing decisions and steer packets accordingly.
- **Marathon**: Unlike Time Multiplexing, Marathon is restricted to using a DOR route. This means we only have freedom in *time* and can delay packet injection at the PE to avoid the congestion hotspot. Thus, our packet will not suffer any deflections like Hoplite, nor be able to take a fastest available route like Time Multiplexing.

C. *Static Scheduling*

In Listing 1, we show the high-level Breadth-First Search (BFS) algorithm that searches for the specific shortest path for a given workload in both space and time. The actual computation is traditional multiple-source multiple-sink Breadth-First search with a modification in the priority queue to hold cost as well as the timestamp of a given segment in the route. For Time Multiplexing, there are no restriction on how the search wavefront proceeds for loop index k (line number 13 in Listing 1) – all unidirectional torus links can be searched in both space and time. For Marathon, the neighborhood expansion in loop k is done in Dimension-Ordered style thereby restricting spatial freedom to strict DOR path. Failure to find a route results in searching for a new route on a delayed starting timeslot. This is also evident in Figure 6, where Time-Multiplexed route is able to take two turns to reach the destination sooner, while Marathon is more constrained and can only avoid congestion by delaying the injection cycle. Additionally, Time Multiplexing can naturally handle fanouts using a single route tree RT in a manner similar to VPR by restarting search for multiple fanouts from existing trees. Hoplite and Marathon NoCs must serially inject a new packet for each fanout. These differences form the basis of potentially superior performance for Time Multiplexing. The benefit of having a cheaper router permits more PEs to be accommodated in a fixed-size FPGA chip, overcoming any potential loss in route quality.

```
Q = priority queue; RT(i) = shortest path
// route each message in workload
for (message i: workload) {
  // search over timeslots
  for (time cyc: start to end) {
    // initialize proprity queue
    PQ.push(source(i),0,cyc);
    while (!found_all_sinks || !PQ.empty()) {
      // get cheapest wire
      <wire,cost,time> j,cost(ij),t = PQ.pop();
      // How fanout is populated differentiates
      // Time-Mux vs. Marathon
      for(wire k: fanout of j) {
        // incr. timeslot and cost by 1 per hop
        PQ.push(k, cost(ij) + 1, t+1);
        if(shortest(k)) {
          backtrace(k)=j;
        }
      }
    }
  }
}
```

Listing 1: Pseudo-code for BFS shortest path search for Time Multiplexing and Marathon NoC scheduling. (not showing backtracking or message fanout handling)

IV. Methodology

A. Hardware Mapping

We compile our NoC hardware blocks, written in Verilog, on the Xilinx Virtex-6 LX240T FPGA device for rapid prototyping. We map the design to the FPGA using Xilinx ISE 14.7 and supply `PBLOCK` floor-planning constraints to generate systems as large as 10×10 switches and PEs. Apart from the Hoplite switch, both Time Multiplexing and Marathon NoCs require a variable component to resource utilization depending on the `Context Memory` size which also affects clock frequency.

We report the resource utilization and frequency data in Table I for a 32b design with N cycles of scheduling time. For the Hoplite router, we do not require any supporting logic overhead in the PE and consequently the `PE LUTs` entry is zero[3] Additionally, the Hoplite PE will require other datapath logic and lightweight handshaking controls. For the time-multiplexed router, we note that the bulk of the variable cost due to context is placed in the router itself. We need a total of

[3]Technically, Hoplite PE does need to provide sufficient storage for sinking all packets, but that requirement is valid for all NoCs in this paper.

TABLE I: Comparing the different NoCs (32b payloads, Xilinx Virtex-6 LX240T). Period is for N=32. 64b SRLs used to store context memories.

NoC	**Router LUTs**	**PE LUTs**	**Router FFs**	**Period (ns)**
Hoplite	60	0	100	2.9
Time-Multiplexed	100 + 4×$\lceil N/64 \rceil$	$\lceil N/64 \rceil$	100	3.4
Marathon	60	$\lceil N/64 \rceil$	100	3.2

2b/mux and 2 outgoing ports for a total of 4b of multiplexer select control. The PE needs a 1b vector to store injection control decision. For the Marathon PE, as expected, we only need to store 1b context in the PE to determine injection time.

B. Workloads and Metrics

Traditionally NoCs are evaluated with synthetically-generated point-to-point workloads [1] with configurable injection rates. Under various traffic patterns and injection rates, the sustained rates are recorded and compared. This is of limited applicability for statically-scheduled networks as (1) we are primarily interested in overcoming latency limits of packet switching, and (2) perform an area-aware performance analysis of the NoCs as they must be mapped to actual FPGAs.

This implies that we must record total completion time for each workload (in addition to sustained rates) and characterize metrics as a function of FPGA LUTs (in addition to PEs). Furthermore, instead of purely relying on statically-generated traffic, we also consider real-world communication traces In this real-world scenario, we cannot vary injection rate directly, but can control total communication workload size and bisection bandwidth. We measure key matrices such as total routing time, injection rate (where applicable), congestion time and network efficiency. We consider two sets of workloads:

- **Statistically-generated point-to-point workloads**: We use standard NoC evaluation workloads including `RANDOM`, `LOCAL`, `TORNADO`, `BITREV` patterns. Each of these datasets are run from 16K packets with each packet having one source and one sink. We do not vary injection rate, but only control total workload size (*i.e.* number of packets to be scheduled) and record the number of cycles required to route the workload. This can be used to calculate sustained rate.
- **Sparse Matrix-Vector Multiplication (SpMV) workloads**: We extract communication graphs from BSP (Bulk-Synchronous Parallel) applications such as sparse matrix-vector multiplication. Here each matrix row is a node in the graph, while non-zeros in each row corresponding to vector-element products become edges in the graph. These communication traces are captured and represented as dataflow graphs where packets can have multiple sink nodes (multiple non-zeros in a matrix columns). Again, we record total number of cycles required to route the workload. Matrices are selected from the Matrix-Market dataset [2].

V. Results

In this section, we discuss resource utilization, and performance results for the different NoCs to understand the engineering tradeoffs. We first evaluate the torus topology against a mesh, provide resource and performance analysis of Marathon.

A. *Choice of Topology*

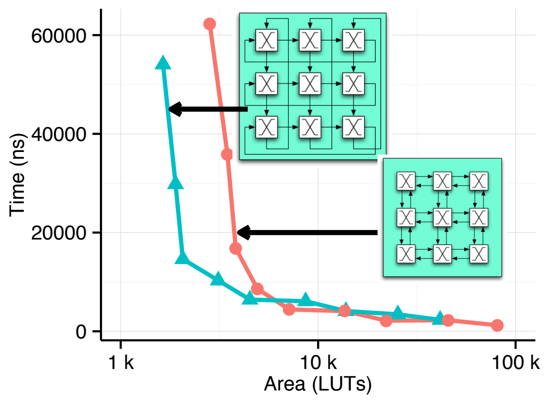


Fig. 7: Time-Multiplexed Routing Topology Selection when injecting 16K packets with `RANDOM` workload. In most cases, Mesh (red) is outperformed by Torus (blue).

Choice of topology has a strong impact on FPGA NoC costs due to complexity of switch implementations and wiring requirements. In Hoplite, the use of a unidirectional torus instead of a bidirectional mesh significantly reduced the cost of the crossbar within the switch and enabled a better match the modern 6-LUT FPGA architecture. The original time-multiplexed FPGA overlay paper [8] used the butterfly fat-tree which was a better fit to the 4-LUT architectures in vogue in that era (Virtex-2 XC2V6000 and Virtex-4 XC4VLX100). However, subsequent studies [10][4] showed the high cost of latencies in the upper stages eliminated any advantages over ordinary bidirectional meshes. This early work did not consider the torus in their analysis. Inspired by Hoplite, we revisit this briefly to evaluate whether a torus-based time-multiplexed NoC delivers competitive performance against a mesh-based time-multiplexed NoC. In Figure 7, we compare the performance of the 2D bidirectional mesh against a torus when routing the `add20` workload (142K packets extracted from sparse matrix-vector multiplication) for various system sizes. We count the area required by the complete NoC (switch multiplexers + context memory + PE context) on a newer Virtex-6 LX240T 6-LUT design. The unidirectional torus delivers cheaper, better-performing NoCs at small system-sizes. At larger system sizes when performance saturates the torus and the mesh have indistinguishable performance. For this paper, we restrict all our comparison to the unidirectional torus-based NoCs.

[4]Figure 6.1, Page 48, *http://thesis.library.caltech.edu/2335/1/final.pdf*.

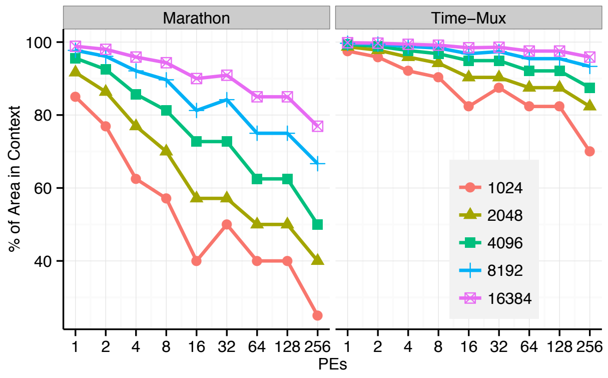


Fig. 8: Impact of Workload Size (colors) on the Cost of Time-Multiplexed and Marathon Switch

B. *Context Overheads*

While we organized the per-switch costs in Table I earlier, we show the effect of size of the workload being mapped on the cost of Time-Multiplexed and Marathon NoCs in Figure 8. Here we calculate overhead as Context LUTs as a % of Total LUTs required. As the Marathon NoC only requires a 1b context to be stored in the PE for injection control, the overheads are 2–3× lower compared to overall NoC costs. For instance, when routing a larger workload requiring 1K cycles, the Marathon NoC overheads are only 30% vs. Time Multiplexing overheads of 70%. These trends generally stay true for other cases evaluated in the paper.

C. *Statistical Patterns*

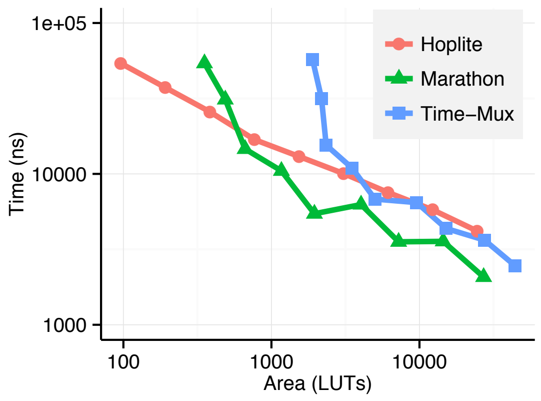


Fig. 9: Comparing NoC Performance for Hoplite, Time Multiplexing and Marathon when routing identical `RANDOM` workload with 16K packets.

Back in Figure 4, we showed how Time Multiplexing outperformed Packet Switching when routing `RANDOM` workload with 16K packets while considering only PE counts (actual cost of PEs and routers not considered) and measuring performance in terms of cycles. In Figure 9, we show the same data by accounting for physical resources costs (x-axis

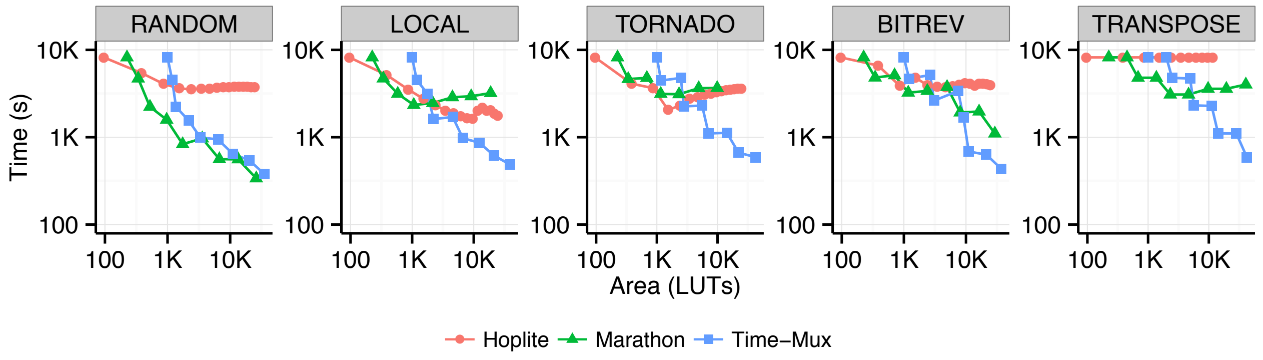


Fig. 10: Evaluating the NoCs with various statistically generated traffic patterns – a total of 16K packets injected for each pattern. For almost all patterns Time Multiplexing delivers substantially better sustained rate ≈0.5. For Marathon NoC, only the `RANDOM` traffic pattern routes particularly well while the rest are 2–3× less capable. Hoplite is beaten in all cases.

is LUTs instead of PEs) and clock frequency (y-axis is time instead of cycles). Now we see that for small system sizes (< 500 LUTs), the ordinary Hoplite-based NoC offers the best runtime (least time required). Above this tiny system size, the Marathon NoC overwhelmingly dominates performance edging out Time Multiplexing entirely. The smaller routers and negligible loss in route quality ensures that Marathon provides the best solution in this space.

In Figure 10, we show the effect of varying area on total routing time (and sustained rates) required to schedule 16K packets under various statistical patterns. Overall we see that the total time taken to route the entire workload generally decreases with more area up to a limit. Broadly, Time Multiplexing delivers better performance than Hoplite in all cases above 2–3K LUTs (≈36–50 PEs) which is expected due to higher quality scheduling. The specific traffic patterns seems to have virtually no impact on time-multiplexed router performance as our greedy router is limited by the scheduling order more strongly than the distribution of packet traversal locations. For Hoplite, performance saturates rather early >1K LUTs with the `LOCAL` pattern showing fastest performance. For Marathon, we see the randomly-generated traffic patterns delivering better performance than locality-based traffic and even exceeding (slightly) the performance of Time Multiplexing. This counter-intuitive observation is linked to the ordering of routes available during static scheduling (line 3 in Listing 1). This effect where randomization performs better than locality-rich traffic has been previously observed in the Connection Machine [14], [6]. The randomization effects allow the static router to avoid pathological worst cases of congestion that are otherwise handled by Hoplite though simple deflections. For most traffic patterns (except `LOCAL` and `TORNADO`), Marathon outperforms Hoplite by 2–10×, but is slower than Time Multiplexing by as much as 2–5× at larger system sizes. For these adversarial traffic scenarios, the restrictions imposed on sequencing routes (line 3 in Listing 1) can be eliminated and a VPR-like multi-pass heuristic deployed to improve performance (future work).

D. *Real-World Traces*

In Figure 11, we show the effect of routing the `add20` sparse matrix-vector workload with 142K messages at various sizes. The Hoplite NoC is only useful at a couple of small system sizes below 500 LUTs. Beyond that the Marathon NoC generally delivers best performance. The Time-Multiplexed NoC catches up and marginally outperforms Marathon only at almost 20× larger system size ≈100K LUTs.

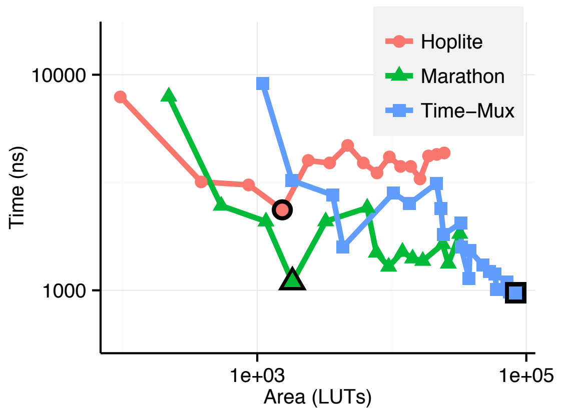


Fig. 11: Comparing Marathon with Hoplite and Time Multiplexing for the 142K message `add20` benchmark. Highlighted data-points correspond to those shown in Figure 1

When considering a range of real-world traces extracted from bulk-synchronous sparse matrix-vector computations, the Marathon NoC deliver better results compared to both Hoplite and Time Multiplexing by 1.3–2.8× (mean 1.4×) when considering best-performing designs. In the case of `add20` benchmark, the static scheduling order as well as DOR restrictions result in very similar performance for time multiplexing and our hybrid design. But, for other benchmarks we evaluate, Marathon easily beats both Hoplite and Time

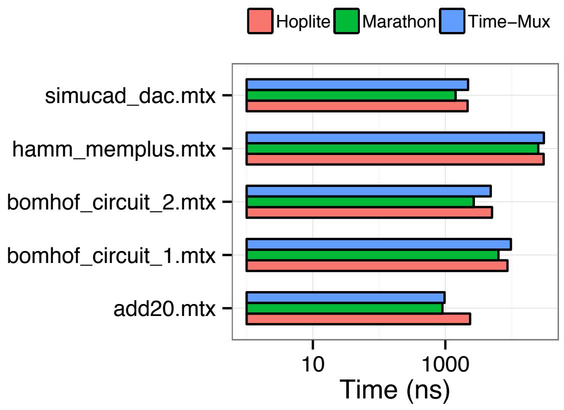


Fig. 12: Comparing NoC routing time for sparse matrix-vector multiplication (best performance achieved).

Multiplexing unlike the case with *synthetic locality* where time multiplexing wins be a large margin at larger system sizes.

VI. Discussion

Time Multiplexing has been explored in the design of video-processing SoCs [9] and high-performance architectures [15] to permit software optimizations to deliver fast implementations of communication while making better use of limited silicon budgets. In contrast, modern FPGAs are much larger, but lightweight time-multiplexed NoCs can still permit developers to use available LUT budgets effectively to parallelize their intended computations. A notion of static reservation of NoC bandwidths was discussed in [12] that allows conventional packet-switched hardware to deliver higher quality of service for critical communication requirements in a NoC. The underlying NoC router hardware used in that study required expensive embodiments of virtual-channel based routers and modifications to the complex flow control schemes to support flit reservation. In contrast, Hoplite packet-switched routers are more amenable for FPGA mapping by using substantially simpler flow control and complete elimination of virtual channel and queuing logic. Furthermore, our time multiplexing enhancements to Hoplite require no modification to actual router hardware. The idea of time-multiplexed NoCs tailored for FPGA fabrics was initially explored in [8] to clearly identify the space of opportunities (application characteristics, injection rates) for time multiplexing when compared to traditional packet switching architectures. Our work shows how to allow higher quality time multiplexing algorithms to work with cheaper packet-switched hardware to go beyond vanilla time multiplexing.

VII. Conclusions

It is possible to build smaller, faster and more efficient FPGA NoC structures by combining (1) packet switching routers which allow fixed-sized implementation of switches, with (2) time-multiplexed packet injection at the processing nodes that enables low latency scheduling of routes. This hybrid approach permits lower overall cycle counts and better congestion management in the FPGA NoC. In this paper, we demonstrate the capabilities of such a hybrid Marathon NoC that is as much as 10× faster than state-of-the-art Hoplite FPGA NoC and 1.2× faster than a Time-Multiplexed FPGA NoC when routing synthetic workloads with the `RANDOM` traffic pattern. For communication traces extracted from real-world traffic patterns, Marathon outperforms both Hoplite and Time Multiplexing by 1.3–2.8×. A key consideration when designing the hybrid routers is the apriori knowledge of the workload which promotes applicability of such designs to sparse linear algebra kernels, signal processing benchmarks or other computational domains with predictable communication patterns. Despite being restricted to deterministic routing algorithms, our space-time router is able to deliver superior performance due to better scheduling optimizations within the restricted search space.

References

[1] P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente, and J. A. Gregorio. TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers. *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 99–106, 2012.

[2] R. F. Boisvert, R. Pozo, K. Remington, R. Barrett, and JJ. The Matrix Market: A web resource for test matrix collections. *Quality of Numerical Software: Assessment and Enhancement*, pages 125–137, 1997.

[3] J. Brassil and R. Cruz. Bounds on maximum delay in networks with deflection routing. *Parallel and Distributed Systems, IEEE Transactions on*, 6(7):724–732, Jul 1995.

[4] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

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

[6] Z. Johan, T. J. Hughes, K. K. Mathur, and S. L. Johnsson. A data parallel finite element method for computational fluid dynamics on the connection machine system. *Computer Methods in Applied Mechanics and Engineering*, 99(1):113–134, 1992.

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

[8] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon. Packet switched vs. time multiplexed FPGA overlay networks. In *Proc. 14th IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 205–216. IEEE, 2006.

[9] A. Laffely, J. Liang, P. Jain, W. Burleson, and R. Tessier. Adaptive systems on a chip (asoc) for low-power signal processing. In *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, volume 2, pages 1217–1221 vol.2, Nov 2001.

[10] N. Mehta. *Time-multiplexed FPGA overlay networks on chip*. PhD thesis, California Institute of Technology, 2006.

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

[12] L.-S. Peh and W. Dally. Flit-reservation flow control. In *High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on*, pages 73–84, 2000.

[13] A. Rohe, S. Teig, H. Schmit, J. Redgrave, and A. Caldwell. Operational time extension, Sept. 8 2009. US Patent 7,587,698.

[14] L. G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.

[15] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring it all to software: Raw machines. *Computer*, 30(9):86–93, Sep 1997.