Routerless Networks-on-Chip

Fawaz Alazemi, Arash Azizimazreah, Bella Bose, Lizhong Chen
Oregon State University, USA
{alazemif, azizimaa, bose, chenliz}@oregonstate.edu

ABSTRACT

Traditional bus-based interconnects are simple and easy to implement, but the scalability is greatly limited. While router-based networks-on-chip (NoCs) offer superior scalability, they also incur significant power and area overhead due to complex router structures. In this paper, we explore a new class of on-chip networks, referred to as *Routerless NoCs*, where routers are completely eliminated. We propose a novel design that utilizes on-chip wiring resources smartly to achieve comparable hop count and scalability as router-based NoCs. Several effective techniques are also proposed that significantly reduce the resource requirement to avoid new network abnormalities in routerless NoC designs. Evaluation results show that, compared with a conventional mesh, the proposed routerless NoC achieves 9.5X reduction in power, 7.2X reduction in area, 2.5X reduction in zero-load packet latency, and 1.7X increase in throughput. Compared with a state-of-the-art low-cost NoC design, the proposed approach achieves 7.7X reduction in power, 3.3X reduction in area, 1.3X reduction in zero-load packet latency, and 1.6X increase in throughput.

1. INTRODUCTION

As technologies continue to advance, tens of processing cores on a single chip-multiprocessor (CMP) has already been commercially offered. Intel Xeon Phi Knight Landing [12] is an example of a single CMP that has 72 cores. With hundreds of cores in a CMP around the corner, there is a pressing need to provide efficient networks-on-chip (NoCs) to connect the cores. In particular, recent chips have exhibited the trend to use many but simple cores (especially for special-purpose many-core accelerators), as opposed to a few but large cores, for better power efficiency. Thus, it is imperative to design highly scalable and ultra-low cost NoCs that can match with many simple cores.

Prior to NoCs, buses have been used to provide on-chip interconnects for multi-core chips [7, 8, 14, 17, 37, 38]. While many techniques have been proposed to improve traditional buses, it is hard for their scalability to keep up with modern many-core processors. In contrast, NoCs offer a decentralized solution by the use of routers and links. Thanks to the switching capability of routers to provide multiple paths and parallel communications, the throughput of NoCs is significantly higher than that of buses. Unfortunately, routers have been notorious for consuming a substantial percentage of chip's power and area [20, 21]. Moreover, the cost of routers increases rapidly as link width increases. Thus, except for a few ad hoc designs, most on-chip networks do not employ link width higher than 256-bit or 512-bit, even though additional wiring resources may be available. In fact, our study shows that, a 6x6 256-bit Mesh only uses 3% of the total available wiring resources (more details in Section 3).

The high overhead of routers motivates researchers to develop *routerless NoCs* that eliminate the costly routers but use wires more efficiently to achieve scalable performance. While the notion of routerless NoC has not been formally mentioned before, prior research has tried to remove routers with sophisticated use of buses and switches, although with varying success. The goal of routerless NoCs is to select a set of smartly placed loops (composed of wires) to connect cores such that the average hop count is comparable to that of conventional router-based NoCs. However, the main roadblocks are the enormous design space of loop selection and the difficulty in avoiding deadlock with little or no use of buffer resources (otherwise, large buffers would defeat the purpose of having routerless NoCs).

In this paper, we explore efficient design and implementation to materialize the promising benefits of routerless NoCs. Specifically, we propose a layered progressive method that is able to find a set of loops that meet the requirement of connectivity and the limitation of wiring resources. The method progressively constructs the design of a large routerless network from good designs of smaller networks, and is applicable to any $n \times m$ many-core chips with superior scalability. Moreover, we propose several novel techniques to address the challenges in designing routerless interface to avoid network abnormalities such as deadlock, livelock and starvation. These techniques result in markedly reduced buffer requirement and injection/ejection hardware overhead. Compared with a conventional router-based Mesh, the proposed routerless design achieves 9.48X reduction in power, 7.2X reduction in area, 2.5X reduction in zero-load packet latency, and 1.73X increase in throughput. Compared with the current state-of-the-art scheme that tries to replace routers with less costly structures (IMR [28]), the proposed scheme achieves 7.75X reduction in power, 3.32X reduction in area,1.26X reduction in zero-load packet latency, and 1.6X increase in throughput.

2. BACKGROUND AND MOTIVATION

2.1 Related Work

Prior work on on-chip interconnects can be classified into *bus-based* and *network-based*. The latter can be further categorized as *router-based NoCs* and *routerless NoCs*. The main difference between bus-based interconnects and routerless NoCs is that bus-based interconnects use buses in a direct, simple and primitive way, whereas routerless NoCs use a network of buses in a sophisticated way and typically need some sort of switching that earlier bus systems do not need. Each of the three categories is discussed in more detail below.

Bus-based Interconnects are centralized communication systems that are straightforward and cheap to implement. While buses work very well for a few cores, the overall performance degrades significantly as more cores are connected to the bus [17, 37]. The two main reasons for such

degradation are the length of the bus and its capacitive load. Rings [7,8,14] can also be considered as variants of bus-based systems where all the cores are attached to a single bus/ring. IBM Cell processor [38] is an improved bus-based system which incorporates a number of bus optimization techniques in a single chip. Despite having a better performance over conventional bus/ring implementations, IBM Cell process still suffers from serious scalability issues [4].

Router-based NoCs are decentralized communication systems. A great deal of research has gone into this (e.g., [10, 13, 18, 23, 25, 26, 31, 33], too many to cite all here). The switching capability of routers provides multiple paths and parallel communications to improve throughput, but the overhead of routers is also quite substantial. Bufferless NoC (e.g., [15]) is a recent interesting line of work. In this approach, buffer resources in a router are reduced to the minimal possible size (i.e. one flit buffer per input port). Although bufferless NoC is a clever approach to reduce area and power overhead, the router still has other expensive components that are eliminated in the routerless approach (Section 7.5 compares the hardware cost).

Routerless NoCs aim to eliminate the costly routers while having scalable performance. While the notion of routerless NoC has not been formally mentioned before, there are several works that try to remove routers with sophisticated use of buses and switches. However, as discussed below, the hardware overhead in these works is quite high, some requiring comparable buffer resources as conventional routers, thus not truly materializing the benefits of routerless NoCs. One approach is presented in [34], where the NoC is divided into segments. Each segment is a bus, and all the segments are connected by a central bus. Segments and central bus are linked by a switching element. In large NoCs, either the segments or the central bus may suffer from scalability issues due to their bus-based nature. A potential solution is to increase the number of wires in the central bus and the number of cores in a segment. However, for NoCs larger than 8×8, it would be challenging to find the best size for the segments and central bus without affecting scalability. Hierarchical rings (HR) [16] has a similar design approach to [34]. The NoC is divided into disjoint sets of cores, and each set is connected by a ring. Such rings are called local rings. Additionally, a set of global rings bring together the local rings. Packets switch between local and global rings through a low-cost switching element. Although the design has many nice features, the number of switching element is still not small. For example, for an 8×8 NoC, there are 40 switching element, which is close to the number of routers in the 8×8 network. Recently, a multi-ring-based NoC called isolated multiple rings (IMR) is proposed in [28] and has been shown to be superior than the above Hierarchical rings. To our knowledge, this is the latest and best scheme so far along the line of work on removing routers. While the proposed concept is promising, the specific IMR design has several major issues and the results are far from optimal, as discussed in the next subsection.

2.2 Need for New Routerless NoC Designs

2.2.1 Principles and Challenges

We use Figure 1 to explain the basic principles of routerless NoCs. This figure depicts an example of a 16-core chip. The 4×4 layout specifies only the positions of the cores, not

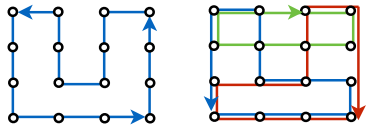

Figure 1: An example of loops in a 4×4 grid.

Table 1: Number of unidirectional loops in $n \times n$ grid [2].

n	# of loops	n	# of loops
1	0	2	2
3	26	4	426
5	18,698	6	2,444,726
7	974,300,742	8	1,207,683,297,862

any topology. A straightforward but naive way to achieve routerless NoC is to use a long loop (e.g., a Hamiltonian cycle) that connects every node on the chip as shown in Figure 1(a). Each node injects packets to the loop and receives packets from the loop through a simple interface (referred to as RL interface hereafter). Apparently, even if a flit on the loop can be forwarded to the next node at the speed of one hop per cycle, this design would still be very slow because of the average $O(n^2)$ hop count, assuming an $n \times n$ many-core chip. Scalability is poor in this case, as conventional topology as such Mesh has an average hop count of $O(n)$.

To reduce the hop count, we need to select a better set of loops to connect the nodes, while guaranteeing that every pair of nodes is connected by at least one loop (so that a node can reach another node directly in one loop). Figure 1(b) shows an example with the use of three loops, which satisfies the connectivity requirement and reduces the all-pair average hop count by 46% compared with (a). Note that, when injecting a packet, a source node chooses a loop that connects to the destination node. Once the packet is injected into a loop, it stays on this loop and travels at the speed of one hop per cycle all the way to the destination node. No changing loops is needed at RL interfaces, thus avoiding the complex switching hardware and per-hop contention that may occur in conventional router-based on-chip networks.

Several key questions can be asked immediately. Is the design in Figure 1(b) optimal? Is it possible to select loops that achieve comparable hop count as conventional NoCs such as Mesh? Is there a generalized method that we can use to find the loops for any $n \times n$ network? How can this be done without exceeding the available on-chip wiring resources? Unfortunately, answering these questions is extremely challenging due to the enormous design space. We calculated the number of possible loops for $n \times n$ chips based on the method used in [2], where a loop can be any unidirectional circular path with the length between 4 and n. Table 1 lists the results up to $n = 8$. As can be seen, the number of possible loops grows extremely rapidly. To make things more challenging, because the task is to find a set of loops, the design space that the routerless NoC approach is looking at is not the number of loops, but the combination of these loops! A large portion of the combinations would be invalid, as not all combinations can provide the connectivity where there is at least one loop between any source and destination pair.

Meanwhile, any selected final set of loops needs to comfortably fit in the available wiring resources on the chip. Specifically, when loops are superpositioned, the number of over-

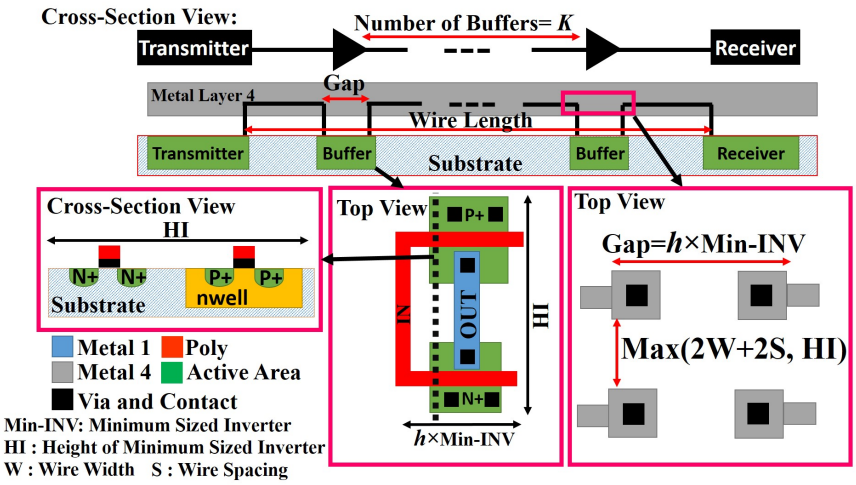

Figure 2: A long wire in NoCs with repeaters.

Table 2: Wiring resources in a many-core processor chip.

Many Core Processor	Xeon Phi, Knights Landing	
Number of Cores	72	
NoC Size	6×6	
Die Area	(31.9mm x 21.4mm) 683 mm^2 [3]	
Technology	FinFET 14nm	
Interconnect	13 Metal Layers	
Inter-core Metal Layers	Metal Layer	Pitch [22] [30]
	M4	80nm
	M5	104nm

lapped loops between any neighboring node pairs should not exceed a limit. In what follows, we use *overlapping* to refer to the number of overlapped loops between two neighboring nodes (e.g., in Figure 1(b) some neighboring nodes have two loops passing through them while others have only one loop passing), and use *overlapping cap* to refer to the limit of the overlapping. Note that the cap should be much lower than the theoretical wiring resources on chip due to various practical considerations (analyzed in Section 3). As an example, if the overlapping cap is 1, then Figure 1(a) has to be the final set. If the overlapping cap increases to 2, it provides more opportunity for improvement, e.g., the better solution in Figure 1(b). The overlapping cap is a hard limit and should not be violated. However, as long as this cap is met, it is actually beneficial to approach this cap for as many neighboring node pairs as possible. Doing this indicates more wires are being utilized to connect nodes and reduce hop count.

2.2.2 *Major Issues in Current State-of-the-Art*

There are several major issues that must be addressed in order to achieve effective routerless NoCs. We use IMR [28] as an example to highlight these issues. IMR is a state-of-the-art design that follows the above principle to deploy a set of rings such that each ring joins a subset of cores. While IMR has been shown to outperform other schemes with or without the use of routers, the fundamental issues in IMR prevent it from realizing the true potential of routerless NoCs. This calls for substantial research on this topic to develop more efficient routerless designs and implementations.

(1) *Large overlapping*. For example, IMR uses a large number of superpositioned rings (equivalent to the above-defined overlapping cap of 16) without analyzing the actual availability of wiring resources on-chip.

(2) *Extremely slow search*. A genetic algorithm is used in IMR to search the design space. This general-purpose search algorithm is very slow (taking several hours to generate results for 16×16, and is not able to produce good results in a reasonable time for larger networks). Moreover, the design generated by the algorithm is far from optimal with high hop counts, as evaluated in Section 6. Thus, efforts are much needed to utilize clever heuristics to speed up the process.

(3) *High buffer requirement*. Currently, the network interface of IMR needs one packet-sized buffer per ring to avoid deadlock. Given that up to 16 rings can pass through an IMR interface, the total number of buffers at each interface is very close to a conventional router.

The above issues are addressed in the next three sections. Section 3 analyzes the main contributing factors that determine the wiring availability in practice, and estimates reasonable overlapping caps using a contemporary many-core processor. Section 4 proposes a layered progressive approach to select a set of loops, which is able to generate highly scalable routerless NoC designs in less than a second (up to 128×128). Section 5 presents our implementation of routerless interface. This includes a technique that requires only one flit-sized buffer per loop (as opposed to one packet-sized buffer per loop). This technique alone can save buffer area by multiple times.

3. ANALYSIS ON WIRING RESOURCES

3.1 Metal Layers

As technology scales to smaller dimensions, it provides a higher level of integration. With this trend, each technology comes with an increasing number of routing metal layers to meet the growing demand for higher integration. For example, Intel Xeon Phi (Knights Landing) [1] and KiloCore [9] are fabricated in the process technology with 11 and 13 metal layers, respectively. Each metal layer has a pitch size which defines the minimum wire width and the space between two adjacent wires. The physical difference between metal layers results in various electrical characteristics. This allows designers to meet their design constraints such as delay on the critical nets by switching between different layers. Typically, lower metal layers have narrower width and are used for local interconnects (e.g., within a circuit block); higher metal layers have wider width and are used for global interconnects (e.g., power supply, clock); middle metal layers are used for semi-global interconnects (e.g., connecting neighboring cores). Table 2 lists several key physical parameters of Xeon Phi including the middle layers that can be used for on-chip networks.

3.2 Wiring in NoC

To estimate the actual wiring resources that can used for

routing, several important issues should be considered when placing wires on the metal layers.

Routing strategy: In general, two approaches can be considered for routing interconnects over cores in NoCs. In the first approach, dedicated routing channels are used to route wires in NoCs. This method of routing was widely used in earlier technology nodes where only three metal layers were typically provided [36], and it has around 20% area overhead. In the second approach, wires are routed over the cores at different metal layers [32]. In the modern technology nodes with six to thirteen metal layers, this approach of routing over logic becomes more common for higher integration. This can be done in two ways: 1) several metal layers are dedicated for routing wires, and 2) a fraction of each metal layer is used to route the wires. The first way is preferable given that many metal layers are available in advanced technology nodes [32, 36].

Repeater: Wires have parasitic resistance and capacitance which increase with the length of wires. To meet a specific target frequency, a long wire needs to be split into several segments, and repeaters (inverters) are inserted between the segments, as shown in Figure 2. The size of repeaters should be considered in estimating the available wiring resources. For a long wire in the NoC, the size of each repeater (h times of an inverter with minimum size) is usually not small, but the number of repeaters (k) needed is small [27]. In fact, it has been shown that increasing K has negligible improvement in reducing the delay [27]. For a 2GHz operating frequency, using only one repeater with the size of 40 times W/L of the minimum sized inverter can support a wire length of 2mm [32], which is longer than the typical distance between two cores in a many-core processor [35].

Coping with cross-talk: Cross-talk noises can occur either between the wires on the same metal layer or between the wires on different metal layers, both of which may affect the number of wires that can be placed. The impact of cross-talk noises on voltage can be calculated by Equation (1) as the voltage changes on a floated victim wire [19].

$$\Delta V_{victim} = \frac{C_{adj}}{C_{victim} + C_{adj}} \times \Delta V_{aggressor} \quad (1)$$

where ΔV_{victim} is the voltage variation on the victim wire, $\Delta V_{aggressor}$ is the voltage variation on the aggressor, C_{victim} is the total capacitance (including load capacitance) of the victim wire, and C_{adj} is the coupling capacitance between the aggressor and the victim. It can be observed from Equation (1) that the impact of cross-talk on the victim wire depends on the ratio of C_{adj} to C_{victim}. Hence, the cross-talk on the same layer has much larger impact on the power, performance, and functionality of the NoC since the adjacent wires which run in parallel on the same metal layer has larger coupling capacitance (C_{adj}) [19]. There are two major techniques to mitigate cross-talk noises, shielding and spacing. In the shielding approach, crosstalk noises are largely avoided between two adjacent wires by inserting another wire (which is usually connected to the ground or supply voltage) between them. In the spacing approach, adjacent wires are separated by a certain distance that would keep the coupling noise below a level tolerable by the target process and application. Compared with spacing, shielding is much more effective as it can almost remove crosstalk noises [5]. However, shielding also incurs more area overhead as the distance used in the spacing approach is usually smaller than that of inserting a wire.

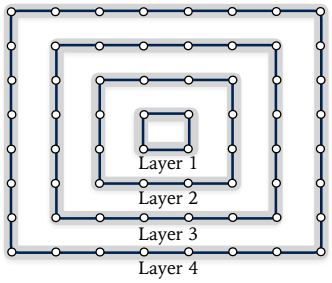

Figure 3: Layers of an 8×8 grid.

3.3 Usable Wires for NoCs

To gain more insight on how many wiring resources are usable for on-chip networks under current manufacturing technologies, we estimated the number of usable wires by taking into account the above factors. The estimation is based on using two metal layers to route wires over the cores. The area overhead of the repeater insertion including the via contacts and the area occupation of the repeaters are considered based on the layout design rules of each metal layer. We used the conservative way of shielding to reduce crosstalk noises (and the inserted wires are not counted towards usable wires), although spacing may likely offer more usable wires. In addition, in practice, 20% to 30% of each dedicated metal layer for routing wires over the cores is used for I/O signals, power, and ground connections [32]. This overhead is also accounted for. The maximum values of h and K are used for worst-case estimation. As such, the above method gives a very conservative estimation of the usable wires. Assuming that there is a chip with similar physical configuration as Table 2, the two metal layers M4 and M5 under 14nm technology can provide 101,520 wires in the cross-section. This translates into 793 unidirectional links of 128-bit, or 396 unidirectional links of 256-bit, or 198 unidirectional links of 512-bit in the cross-section. In contrast, a 6×6 mesh only uses 12 unidirectional 256-bit links in the bisection, which is about 3% of the usable wires. It is important to note that the conventional router-based NoCs do not use very wide links for good reasons. For instance, router complexity (e.g., the number of crosspoints in switches, the size of buffers) increases rapidly as the link width increases. Also, although wider links provide higher throughput, it is difficult to capitalize on wider links for lower latency. The reduction in serialization latency by using wider links quickly becomes insignificant as link width approaches the packet size. This motivates the need for designing routerless NoCs where wiring resources can be used more efficiently.

The above estimation of the number of usable wires helps to decide the overlapping cap mentioned previously. To avoid taxing too much on the usable wiring resources and to have a scalable design, we propose to use an overlapping cap of n for $n \times n$ chips. In the above 6×6 case, this translates into 4.5% of the usable wires for 128-bit loop width, or 9.1% for 256-bit loop width. This parameterized overlapping cap helps to provide the number of loops that is proportional to chip size, so the quality of the routerless designs can be consistent for larger chips.

4. DESIGNING ROUTERLESS NOCS

4.1 Basic Idea

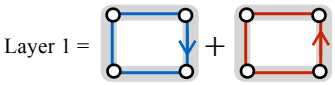

Figure 4: Loops in L_1, and $M_2 = L_1$.

Our proposed routerless NoC design is based on what we call *layered progressive* approach. The basic idea is to select the loop set in a progressive way where the design of a large routerless network is built on top of the design of smaller networks. Each time the network size increments, the newly selected loops are conceptually bundled as a layer that is reused in the next network size.

Specifically, let M_k be the final set of selected loops for $k \times k$ grid ($2 \leq k \leq n$) that meets the connectivity, overlapping and low hop count requirements. We construct M_{k+2} by combining M_k with a new set (i.e., layer) of smartly placed loops. The new layer utilizes new wiring resources that are available when expanding from $k \times k$ to $(k+2) \times (k+2)$. The resulting M_{k+2} can also meet all the requirements and deliver superior performance. For example, as shown in Figure 3, the grid is logically split into multiple layers with increasing sizes. Let L_k be the set of loops selected for Layer k. Firstly, suppose that we already find a good set of loops for 2×2 grid that connects all the nodes with a low hop count and does not exceed an overlapping of 2 between any neighboring nodes. That set of loops is M_2, which is also L_1 as this is the base case. Then we find another set of loops L_2, together with M_2, can form a good set of loops for 4×4 grid (i.e., $M_4 = L_2 \cup M_2$). The resulting M_4 can connect all the nodes with a low hop count and do not exceed an overlapping of 4 between any neighboring nodes. And so on so forth, until reaching the targeted $n \times n$ grid. In general, we have $M_n = L_{\lfloor n/2 \rfloor} \cup M_{n-2} = L_{\lfloor n/2 \rfloor} \cup L_{\lfloor n/2 \rfloor - 2} \cup M_{n-4} = \ldots = L_{\lfloor n/2 \rfloor} \cup L_{\lfloor n/2 \rfloor - 2} \cup L_{\lfloor n/2 \rfloor - 4} \cup \ldots \cup L_1$.

Apparently, the key step in the above progressive process is how to select the set of loops in Layer k, which enables the progression to the next sized grid with low hop count and overlapping. In the next subsections, we walk through several examples to illustrate how it is done to progress from 2×2 grid to 8×8 grid.

4.2 Examples

4.2.1 2×2 *Grid*

This is the base case with one layer. There are exactly two possible loops, one in each direction, in a 2×2 grid. Both of them are included in $M_2 = L_1$, as shown in Figure 4. The resulting M_2 satisfies the requirement that every source and destination pair is connected by at least one loop. The maximum number of loops overlapping between any neighboring nodes is 2, which meets the overlapping cap. This set of loops achieves a very low all-pair average hop count of 1.333, which is as good as the Mesh.

4.2.2 4×4 *Grid*

M_4 consists of loops from two layers. Based on our layered progressive approach, L_1 is from M_2. We select 8 loops to form L_2, as illustrated in Figure 5. The 8 loops fall into four groups (from this network size and forward, each new layer is constructed using four groups with the similar heuristics as discuss below). The first group, A_4 (the subscript indicates the size of the grid), has only one anti-clockwise loop. It provides connectivity among the 12 new nodes when expanding from Layer 1 to Layer 2. The loops in the second group, B_4, have the first column as the common edge of the loops, but the opposite edge of the loops moves gradually towards the right (this is more evident in group B_6 in Figure 6). Similarly, the third group, C_4, uses the last column as the common edge of the loops and gradually moves the opposite edge towards the left. It can be verified that groups B_4 and C_4 provide connectivity between the 12 new nodes in Layer 2 and the 4 nodes in Layer 1. Since the connectivity among the 4 inner nodes has already been provided by L_1, the connectivity requirement of 4×4 grid is met by having L_1, A_4, B_4 and C_4. The fourth group, D_4, offers additional "shortcuts" in the horizontal dimension.

A very nice feature of the selected M_4 is that the wiring resources are efficiently utilized, as the overlapping between many neighboring node pairs is close to the overlapping cap of 4. For example, for the first (or the last) column, each group of loops has exactly one loops passing through that column, totaling an overlapping of 4, which is the same as the cap. Thus, no overlapping "ration" is under-utilized. For the second column (or the third) column, groups A_4 and D_4 have no loop passing through, and groups B_4 and C_4 have two loops passing through in total. However, note that the final M_4 also includes L_1 which has two loops passing through the second (or the third) column. Hence, the total overlapping of the middle columns is also 4, exactly the same as the cap. Simple counting can show that the overlapping on the horizontal dimension is also 4 for each row. Owing to this efficient use of wiring resource "ration", the all-pair average hop count is 3.93 for the selected set of loops in M_4. The final set is $M_4 = L_2 \cup M_2 = L_2 \cup L_1$.

4.2.3 6×6 *Grid*

M_6 consists of loops from three layers. L_1 and L_2 are from M_4, and L_3 is formed in a similar fashion as 4×4 grid from four groups, as illustrated in Figure 6. Again, connectivity is provided by M_4 and groups A to C. Together with group D, the number of overlapping on each column and row is 6, thus fully utilizing the allocated wiring resources.

Additionally, for the purpose of reducing hop count and balancing horizontal and vertical wiring utilization, when we combine M_4 and L_3 to form M_6, every loop in M_4 is reversed and then rotated for 90° clockwise[1]. If this slightly changed M_4 is denoted as M_4', the final set can be expressed as $M_6 = L_3 \cup M_4' = L_3 \cup (L_2 \cup L_1)'$, with an all-pair average hop count of 6.07.

4.2.4 8×8 *Grid*

Similar to earlier examples, L_4 consists of loops shown in Figure 7. The final set M_8 is $M_8 = L_4 \cup M_6' = L_4 \cup \big(L_3 \cup (L_2 \cup L_1)'\big)'$ with an all-pair average hop count of 8.32.

4.3 Formal Procedure

For an $n \times n$ grid, the loops for a routerless NoC design can be recursively found by the procedure shown in Algorithm 1. The procedure is recursive and denoted as RLrec. The procedure begins by generating loops for the outer layer, say layer i, and then it recursively generates loops for layer $i-1$

[1] In 4×4 grid, reversal and rotation of M_2 is not necessary because M_2 and M_2' have the same effect on L_1.

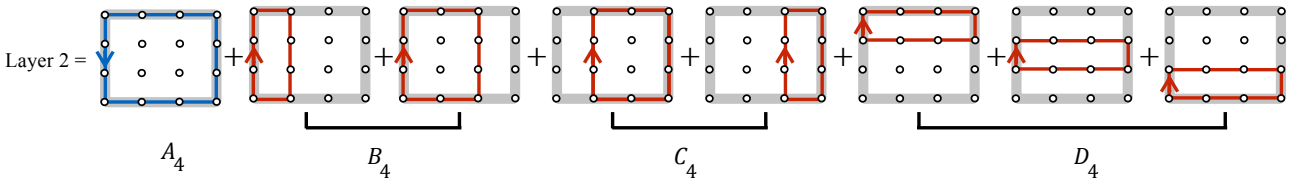

Figure 5: Loops in L_2. $M_4 = L_2 \cup L_1$.

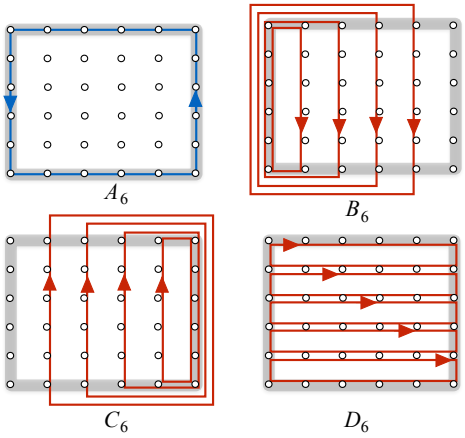

Figure 6: Loops in L_3. $M_6 = L_3 \cup L_2 \cup L_1$.

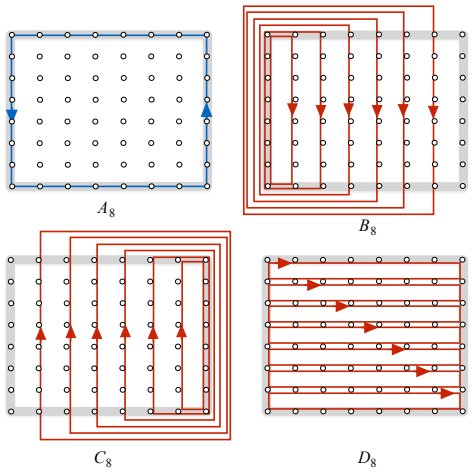

Figure 7: Loops in L_4. $M_8 = L_4 \cup L_3 \cup L_2 \cup L_1$.

Algorithm 1: RLrec

Input : N_L, N_H; the low and high numbers
begin
  if $N_L = N_H$ **then**
    return {}
  Let $M = \{\}$
  if $N_H - N_L = 1$ **then**
    $M = M \cup G(N_L, N_H, N_L, N_H$, clockwise)
    $M = M \cup G(N_L, N_H, N_L, N_H$, anticlockwise)
    return M
  $M = M \cup G(N_L, N_H, N_L, N_H$, anticlockwise) // Group A
  for $i = N_L + 1 \rightarrow N_H - 1$ **do**
    $M = M \cup G(N_L, N_H, N_L, i$, clockwise) // Group B
    $M = M \cup G(N_L, N_H, i, N_H$, clockwise) // Group C
  for $i = L \rightarrow H - 1$ **do**
    $M = M \cup G(i, i+1, N_L, N_H$, clockwise) // Group D
  $M' =$ RLrec(N_L+1, N_H-1)
  Reverse and rotate for 90° every loop in M'
  return $M \cup M'$

and so on until the base case is reached or the layer has a single node or empty. Procedure $G(r_1, r_2, c_1, c_2, d)$ is a simple function that generates a rectangular shape loop with corners (r_1, c_1), (r_1, c_2), (r_2, c_1) and (r_2, c_2) and direction d. When processing each layer in this algorithm, procedure G is called repeatedly to generate four groups of loops. Additionally, the generated loops rotate 90 degrees and reverse directions after processing each layer to balance wiring utilization and reduce hop count, respectively. The final loops generated by the RLrec algorithm have an overlapping of at most n.

While it would be ideal if an analytical expression can be derived to calculate the average hop count for this heuristic approach, this seems to be very challenging at the moment. However, it is possible to calculate the average hop count numerically. This result is presented in the evaluation, which shows that our proposed design is highly scalable.

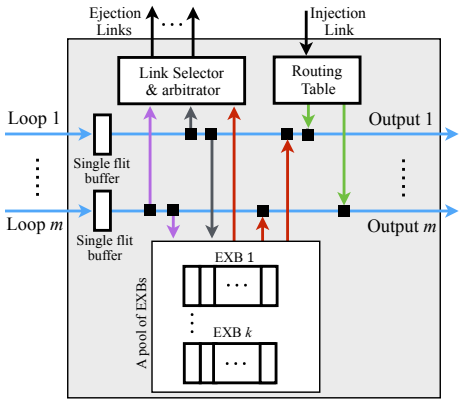

Figure 8: Routerless interface components.

5. IMPLEMENTATION DETAILS

After addressing the key issue of finding a good set of loops, the next important task is to efficiently implement the routerless NoC design in hardware. Because of the routerless nature, no complex switching or virtual channel (VC) structure is needed at each hop, so the hardware between nodes and loops has a small area footprint in general. However, due to various potential network abnormalities such as deadlock, livelock, and starvation, a certain number of resources are required to guarantee correctness. If not addressed appropriately, this may cause substantial overhead that is comparable to router-based NoCs. In this section, we propose a few

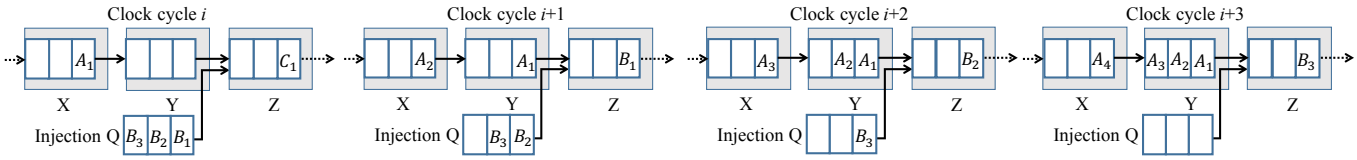

Figure 9: Injecting a long packet requires a packet-sized buffer per loop at each hop in prior implementation (X, Y and Z are interfaces).

effective techniques to minimize those overhead.

In a routerless NoC, each node uses an interface (RL interface) to interact with one or multiple loops that pass through this node. Figure 8 shows the main components of a RL interface. While details are explained in the following subsections, the essential function of the interface includes injecting packets into a matching loop based on connectivity and availability, forwarding packets to the next hop on the same loop, and ejecting packets at the destination node. Notice that packets cannot switch loops once injected. All the loops have the same width (e.g., 128-bit wires).

5.1 Injection Process

5.1.1 Extension Buffer Technique

A loop is basically a bundle of wires connected with flip-flops at each hop (Figure 8). At clock cycle i, a flit arriving at the flip-flop of loop l must be consumed immediately by either being ejected at this node or forwarded to the next hop on loop l through output l. If no flit arrives at loop l (thus not using output l), the RL interface can inject a new flit on loop l through output l. However, it is possible that an injecting packet consists of multiple flits and requires several cycles to finish the injection, during which other flits on loop l may arrive at this RL interface. Therefore, addition buffer resources are needed to hold the incoming flits temporarily.

If routerless NoC uses the scheme proposed in prior ring-based work (e.g., IMR [28]), a full packet-sized buffer per loop at each hop would be needed to ensure correctness, which is very inefficient. As illustrated in Figure 9, a long packet B with multiple flits is waiting for injection (there is no issue if it is a short single-flit packet). At clock cycle i, the injection is allowed because packet B sees that no other flit in Interface Y is competing with B for the output to Interface Z. From cycle $i+1$ to $i+3$, the flits of B are injected sequentially. However, while packet B is being injected during these cycles, another long packet A may arrive at Interface Y. Because RL interfaces do not employ flow control to stop the upstream node, Interface Y needs to provide a packet-sized buffer to temporarily store the entire packet A. A serious inefficiency lies in the fact that, if there are m loops passing through a RL interface, the interface needs to have m packet-sized buffers, one for each loop.

To address this inefficiency, we notice that an interface injects packets one at a time, so not all the loops are affected simultaneously. Based on this observation, we propose the extension buffer technique to share the packet-sized buffer among loops. As shown in Figure 8, each loop has only a flit-sized buffer, but the interface has a pool of extension buffers (EXBs). The size of each EXB is the size of a long packet, so when a loop is "extended" with an EXB, it would be large enough to store a long packet. Minimally, only one EXB is needed in the pool, but having multiple EXBs may have slight performance improvement. This is because another injection might occur while the previous EXB is not entirely released (drained) due to a previous injection (e.g., clock cycle $i+3$ in Figure 9). However, as shown later in the evaluation, the performance difference is negligible. As a result, our proposed technique of using one shared EXB can essentially achieve the same objective of ensuring correctness as IMR but reduces the buffer requirement by m times. This is equivalent to an 8X saving in buffer resources in 8×8 networks and 16X saving in 16×16 networks.

5.1.2 Injection Process

The injection process with the use of EXBs is straightforward. To inject a packet p of n_f flits, the first step is to look up a small routing table to see which loop can reach p's destination. The routing table is pre-computed since all the loops are pre-determined. The packet p then waits for the loops to become available (i.e., having sufficient buffer space). Assume l is a loop that has the shortest distance to the destination among all the available loops. When the injection starts, the interface holds the output port of l for n_f cycles to inject p, and assigns a free EXB to l if $n_f > 1$ and l is not already connected to another EXB. During those n_f cycles, any incoming flit through the input port of l is enqueued in the extension buffer. The EXB is released later when its buffer slots are drained.

5.2 Ejection Process

The ejection process starts as soon as the head flit of a packet p reaches the RL interface of its destination node. The interface ejects p, one flit per cycle. Once p is ejected, the interface will wait for another packet to eject. There is, however, a potential issue with the ejection process. While unlikely, a RL interface with m loops may receive up to m head flits simultaneously in a given cycle that are all destined to this node. Because any incoming packets need to be consumed immediately and the packets are already at the destination, the interface needs to have m ejection links in order to eject all the packets in that cycle. As each eject link has the same width as the loop (i.e., 128-bit), this incurs substantial hardware overhead.

To reduce this overhead, we utilize the fact that the actual probability of having k packets ($1 < k \leq m$) arriving at the same destination in the same cycle is low, and this probability decreases drastically as k increases. Based on this observation, we propose to optimize for the common case where only e ejection links are provided ($e \ll m$). If more than e packets arrive at the same cycle, $(k-e)$ packets are forwarded to the next hop. Those deflected packets will continue on their respective loops and will circle back to the destination later. As shown in the evaluation, having two ejection links can reduce the percentage of circling packets to be below 1% on average (1.6% max) across the benchmarks. This demonstrates that this is a viable and effective technique to reduce overhead.

5.3 Avoiding Network Abnormalities

As network abnormalities are theoretically possible but

practically unlikely scenarios, our design philosophy is to place very relaxed conditions to trigger the handling procedures, so as to minimize performance impact while guaranteeing correctness.

5.3.1 Livelock Avoidance

A livelock may occur if a packet circles indefinitely and never gets a chance to eject. We address this issue by having a byte-long circling counter at each head flit with an initial value of zero. Every time a packet reaches its destination interface and is forced to be deflected, the counter is incremented by 1. If the circling counter of a packet p reaches 254 but none of the ejection link is available, the interface marks one of its ejection links as reserved and then deflects p for the last time. The marked ejection link will not eject any more packets after finishing the current one, until p circles back to the ejection link (by then the marked ejection link will be available; otherwise there is a possible protocol-level deadlock, discussed shortly). Once p is ejected, the interface will unmark the ejection link for it to function normally. Due to the extremely low circling percentage (maximum 3 times of circling for any packet in our simulations), this livelock avoidance scheme has minimal performance impact.

5.3.2 Deadlock Avoidance

With no protocol-level dependence at injection/ejection endpoints, routing-induced deadlock is not possible in routerless NoCs as packets arriving at each hop are either ejected or forwarded immediately. Hence, a packet can always reach its destination interface without being blocked by other packets. The above livelock avoidance ensures that the packet can be ejected within a limited number of circlings.

With more than one dependent packet types, the marked ejection link in the above livelock avoidance scheme may not able to eject the current packet (say a request packet) in the ejection queue, because the associated cache controller cannot accept new packets from the ejection queue (i.e., input of the controller). This may happen when the controller itself is waiting for packets (say a reply packet) in the injection queue (i.e., output of the controller) to be injected into the network. A potential protocol-level deadlock may occur if that reply packet cannot be injected, such as the loop is full of request packets that are waiting to be ejected.

To avoid such protocol-level deadlock, the conventional approach is to have a separate physical or virtual network for each dependent packet type. While similar approach can be used for routerless NoCs, here we propose a less resource demanding solution. which is made possible by the circling property of loops. This solution only needs an extra reserved EXB, as well as a separate injection and ejection queue for each dependent packet type. The separate injection/ejection queues can come from duplicating original queues or from splitting the original queues to multiple queues. In either case, the loops and wiring resources are *not* duplicated, which is important to keep the cost low. Following the above livelock avoidance scheme, when a packet p on loop l completes the final circling (counter value of 255) and finds that the marked ejection link is still not available, p is temporarily buffered in the reserved EXB instead of forwarding to output l. Meanwhile, we allow the head packet q in the injection queue of the terminating packet type (e.g., a reply packet in the request-reply example) to inject into loop l through output l. Once q is injected, the cache controller is able to put another reply packet in its output (i.e., the injection queue) which, in turn, allows the controller to accept a new request from its input (i.e., the ejection queue). This creates space in the ejection queue to accept packet p that is previously stored in the reserved EXB. Once p moves to the ejection queue, the EXB is freed. Essentially, the reserved EXB acts as a temporary exchanging space while the separate injection/ejection queues avoid blocking of different packet types at the endpoints.

5.3.3 Starvation Avoidance

The last corner case we address is starvation. With the previous livelock and deadlock handling, if a packet is consumed at its destination RL interface, the interface can use the free output to inject a new packet. However, it is possible that a particular interface X is not the destination of any packets and there is always a flit passing through X every single cycle. This never occurred in any of our experiments as it is practically impossible that a cache bank is not accessed by any other cores. However, it is theoretically possible and, when occurred, prevents X from injecting new packets. We propose the following technique to avoid starvation for the completeness of the routerless NoC design. If X cannot inject a packet after a certain number of clock cycles (a very long period, e.g., hundreds of thousand cycles or long enough to have negligible impact on performance), X piggybacks the next passing head flit f with the ID of X. When f is ejected at its destination interface Y, instead of injecting a new packet, Y injects a single-flit no-payload dummy packet that is destined to X. When the dummy packet arrives at X, X can now inject a new packet by using the free slot created by the dummy packet. This breaks the starvation configuration.

5.4 Interface Hardware Implementation

Figure 8 depicts the main components of a RL interface. We have explained the extension buffers (EXBs), single-flit buffers, routing table, and multiple ejection links in the previous subsections. The arbitrator receives flits from input buffers and selects up to e input loops for ejection based on the oldest first policy. The arbitrator contains a small register that holds the arbitration results. The link status selector is a simple state machine associated with the loops. It monitors the input loops and arbitration results, and changes the state of the loops (e.g., ejection, stall in extension buffers, etc.) in the state machine. There are several other minor logic blocks that are not shown in Figure 8 for better clarity. Note that the RL interface does not use the information of neighboring nodes, which differs from most conventional router-based NoCs that need credits or on/off signals for handshaking.

To ensure the correctness of the proposed interface hardware, we implement the design in RTL Verilog that includes all the detailed components. The Verilog implementation is verified in Modelsim, synthesized in Synopsys Design Compiler, and placed and routed using Cadence Encounter tool. We use the latest 15nm process NanoGate FreePDK 15 Cell Library [29] for more accurate evaluation. As a key result, the RL interface is able to operate at up to 4.3GHz frequency while keeping the packet forwarding process in one clock cycle. This is fast enough to match up with most commercial many-core processors. Injecting packets may take an additional cycle for table look-up. In the main evaluation below, both the interfaces and cores are operating at 2GHz.

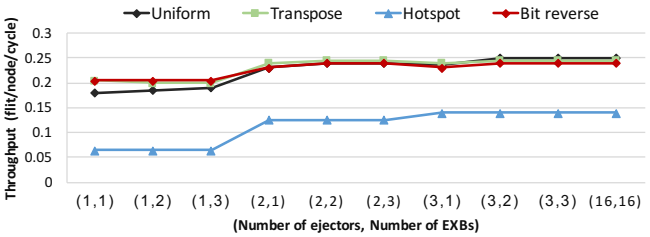

Figure 10: Throughput of routerless NoC under different number of ejection links and extension buffers (EXBs).

6. EVALUATION METHODOLOGY

We evaluate the proposed routerless NoC (RL) extensively against Mesh, EVC, and IMR in Booksim [24]. For synthetic traffic workloads, we use uniform, transpose, bit reverse, and hotspot (with 8 hotspots nodes). BookSim is warmed up for $10,000$ clock cycles and then collects performance statistics for another $100,000$ cycles at various injection rates. The injection rate starts at 0.005 flit/node/cycle and is incremented by 0.005 flit/node/cycle until the throughput is reached. Moreover, we integrate Booksim with Synfull [6] for performance study of PARSEC [11] and SPLASH-2 [39] benchmarks. Power and area studies are based on Verilog post-synthesis simulations, as described in Section 5.4.

In the synthetic study, each router in Mesh is configured with relatively small buffer resources, having 2 VCs per link and 3 flits per VC. The link width is set to 256-bit. Also, the router is optimized with lookahead routing and speculative switch allocation to reduce pipeline stages to 2 cycles per router and 1 cycle per link. EVC has the same configuration as Mesh except for one extra VC that is required to enable express channels. For IMR, the ring set is generated by the evolutionary approach described in [28]. To allow a fair comparison with RL, the maximum number of overlapping cap, for both RL and IMR, is set to n for $n \times n$ NoC. We also follow the original paper to faithfully implement IMR's network interface. Each input link in an IMR's interface is attached with a buffer of 5 flits and the link width is set to 128-bit (the same as the original paper). In RL, loops are generated by RLrec algorithm and accordingly the routing table for each node is calculated. Each interface is configured with two ejection links and each input link has a flit-size buffer. Also, an EXB of 5 flits is implemented in each interface. The link width is 128-bit (the same as IMR). In all the designs, packets are categorized into data and control packets where each control packet has 8 bytes and each data packet has 72 bytes. Accordingly, data packets in Mesh, EVC, IMR, and RL are of 3, 3, 5 and 5 flits, respectively, and the control packets are of a single flit.

For benchmark performance study, we also add 2D Mesh with various configurations as well as a 3D Cube design into the comparison. RL has the same configuration as the synthetic study. For 2D Mesh, we use 9 configurations, each having the configuration M(x,y) where $x \in \{1,2,3\}$ is the router delay and $y \in \{1,2,3\}$ is the buffer size, i.e., routers with 1-cycle, 2-cycle and 3-cycle delay, and with 1-flit, 2-flit and 3-flit buffer size. 3D Cube is configured with 2 VCs per link, 3 flits per VC, and 2-cycle per hop latency.

7. RESULTS AND ANALYSIS

7.1 Ejection Links and Extension Buffers

The proposed RL scheme is flexible to use any number of ejection links and EXBs. On the ejection side, the advantages of having more ejection links are higher chance for packet ejection and lower chance for packet circling in a loop. However, adding more ejection links complicates the design of the interface and leads to additional power and area overhead in the interface and the receiving node. On the injection side, EXBs have a direct effect on the injection latency of long packets. Recall that, a loop must be already attached with an EXB or a free EXB is available to be able to inject a long packet. Similar to ejection links, having more EXBs can lower injection latency but incur larger area and power overhead.

We studied the throughput of RL with different configurations of ejection links and EXBs on various synthetic traffic patterns. The NoC size for this study is 8×8. The results are shown in Figure 10. In the figure, each configuration is denoted by (x,y) where x is the number of ejection links and y is the number of EXBs. The basic and best in terms of area and power overhead is $(1,1)$ configuration but it has the worst performance. By adding up to three EXBs with a single ejection link, the throughput is only slightly changed (less than 5%). This indicates that the number of EXBs is not very critical to performance, and it is possible to use only one EXB for injecting long packets while saving buffer space.

For $(2,1)$ configuration, it doubles the chance for packet ejection when compared to $(1,x)$ configurations. The throughput is notably improved by an average of 38% for all the patterns when compared to $(1,1)$ configurations. For instance, hotspot traffic pattern has 0.125 throughput in $(2,1)$ configuration but only 0.065 in $(1,1)$, a 92.5% improvement). However, on top of $(2,1)$ configuration, adding up-to three EXBs (i.e., $(2,3)$) improves throughput only by 5% on average.

Given all the results, we choose the $(2,1)$ configuration as the best trade-off point, and use it for the remainder of this section. We also plot the $(16,16)$ configuration which is the ideal case (no blocking in injection or ejection may happen). As can be seen, $(2,1)$ is very close to the ideal case. Section 7.3 provides a detailed study for the number of times packet circling in loops for the $(2,1)$ configuration.

7.2 Synthetic Workloads

Figure 11 plots the performance results of four synthetic traffic patterns for an 8×8 NoC. RL has the lowest zero-load packet latency in all four traffic patterns. For example, in uniform random, the zero-load packet latency is 21.2, 14.9, 10.5, and 8.3 cycles for Mesh, EVC, IMR, and RL, respectively. When averaged over the four patterns, RL has an improvement of 1.59x, 1.43x, and 1.25x over Mesh, EVC, and IMR, respectively. RL achieves this due to low per hop latency (one cycle) and low hop count.

In terms of throughput, the proposed RL also has advantage over other schemes. For example, the throughput for hotspot is 0.08, 0.05, 0.06, and 0.125 (per flit/node/cycle) for Mesh, EVC, IMR, and RL, respectively. In fact, RL has the highest throughput for all the traffic patterns. When averaged over the four patterns, RL improves throughput by 1.73x, 2.70x, and 1.61x over Mesh, EVC, and IMR, respectively. This is mainly owing to the better utilization of wiring resources in RL. Note that, EVC has a lower throughput than Mesh as EVC is essentially a scheme that trades off throughput for lower latency at low traffic load.

7.3 PARSEC and SPLASH-2 Workloads

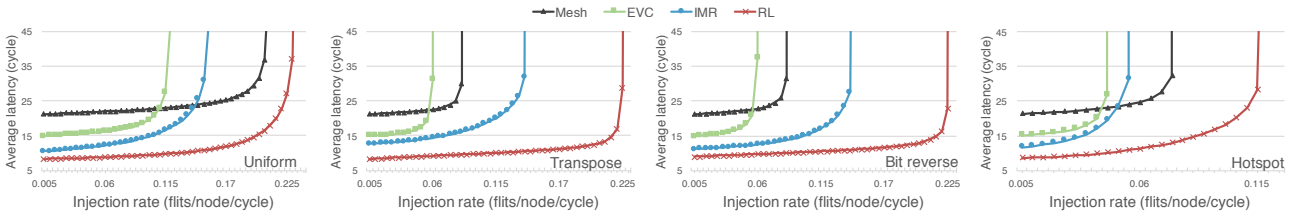

Figure 11: Performance comparison for synthetic traffic patterns.

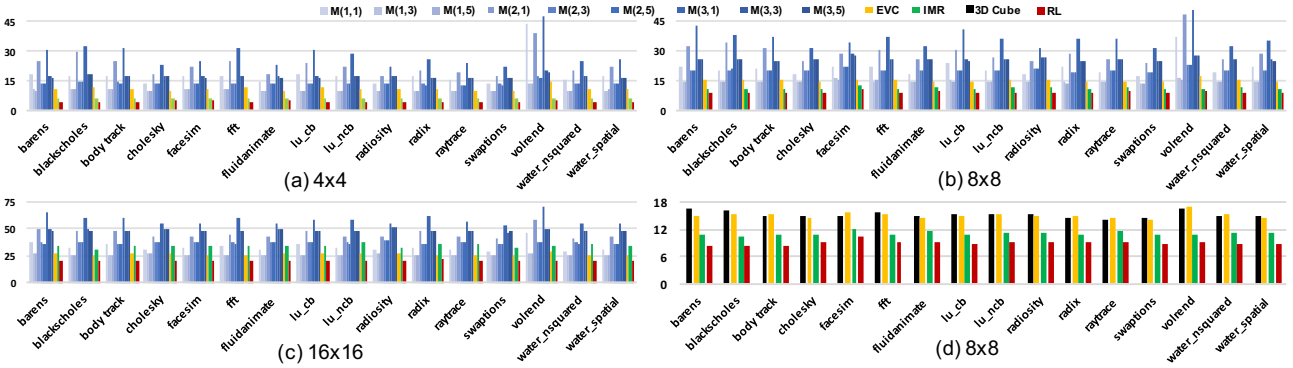

Figure 12: PARSEC and SPLASH-2 benchmark performance results (y-axis represents average pack latency in cycles.) RL is compared with different Mesh configurations, EVC, and IMR in (a), (b) and (c). In (d), RL is also compared with a 3D Cube.

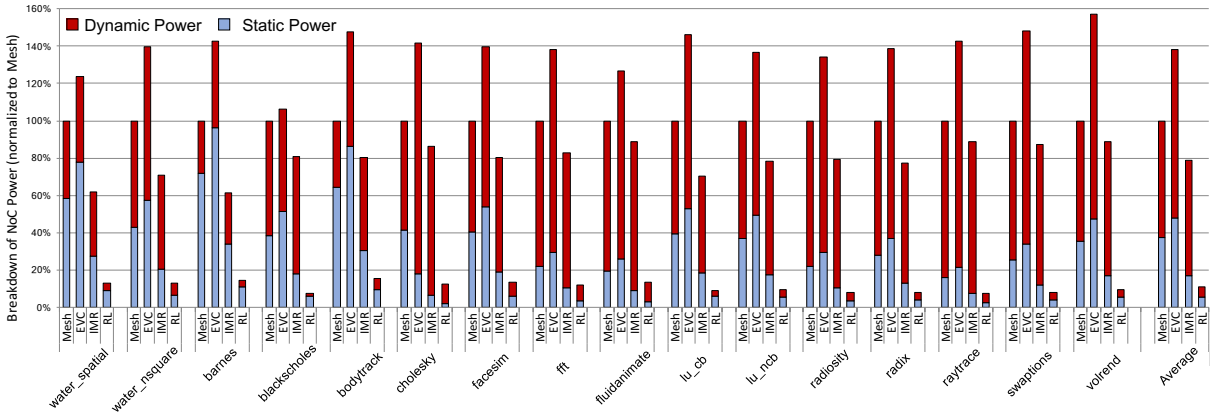

Figure 13: Breakdown of power consumption for different PARSEC and SPLASH-2 workloads (normalized to Mesh).

We utilize Synfull and Booksim to study the performance of RL, 2D Mesh with different configurations, EVC, IMR, and a 3D Cube under 16 PARSEC and SPLASH-2 benchmarks. The NoC sizes under evaluation are 4×4, 8×8 and 16×16 for RL, 2D Mesh, EVC and IMR, and $4 \times 4 \times 4$ for 3D cube. Figure 12 shows the results.

In Figure 12(a)-(c), RL is compared against 2D Mesh, EVC and IMR. From the figures, the best configuration for Mesh is M(1,5) (i.e. per hop latency of 1 and buffer size of 5) and the worst is M(3,1). Lowering per hop latency in Mesh helps to improve overall latency, and reducing buffer sizes may cause packets to wait longer for credits and available buffers. The average packet latency of RL in 4×4, 8×8, and 16×16 are 4.3, 8.9 and 20.1 cycles, respectively. This translates into an average latency reduction of RL over M(1,5) by 57.8%, 38.4% and 22.2% in 4×4, 8×8 and 16×16, respectively. The IMR rings in 16×16 are very long and seriously affects its latency. RL reduces the average latency by 23.3% over EVC and 41.2% over IMR.

In Figure 12(d), the performance of 3D cube is clearly better than all the Mesh configurations in (b) mainly due to lower hop count and larger bisection bandwidth. Despite this, RL still offers better performance than 3D cube. The average latency of RL is 8.9 cycles, which is 41% lower than the 15.2 cycles of 3D cube.

7.4 Power

Figure 13 compares the power consumption of Mesh (i.e. M(2,3)), EVC, IMR and RL for different benchmarks, normalized to the Mesh. All the power consumption shown in this Figure are reported after P&R in NanGate FreePDK 15 Cell Library [29] by Cadence Encounter. The activity factors for the power measurement are obtained from Booksim, and the power consumption includes that of all the wires.

The average dynamic power consumption for RL is only 0.26mW, and for Mesh, EVC and IMR the average is 2.88mW, 4.27mW and 2.91mW, respectively. Because RL has no crossbar, it requires only 9%, 6.1% and 8.9% of the dynamic power consumed by Mesh, EVC and IMR, respectively. Meanwhile,

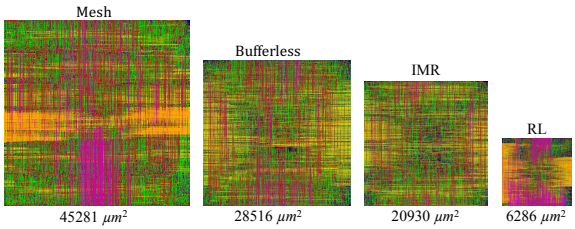

Figure 14: Area comparison under 15nm technology.

static power is mostly consumed by buffers. Unlike Mesh, EVC and IMR, RL has a much lower buffer requirement. As a result, RL consumes very low static power of 0.18mW on average, while Mesh, EVC and IMR consume 1.39mW, 1.64mW and 0.58mW, respectively. Adding dynamic and static power together, on average, RL reduces the total NoC power consumption by 9.48X, 13.1X and 7.75X over Mesh, EVC and IMR, respectively.

7.5 Area

Figure 14 compares the router or interface area of the different schemes we are studying. The results are obtained from Cadence Encounter after P&R[2]. We also add a bufferless design to the comparison. The largest area is 60731μm^2 for EVC (not shown in the figure) followed by 45281μm^2, 28516μm^2, 20930μm^2 and 6286μm^2 for Mesh, Bufferless, IMR and RL, respectively. The EXB and ejection link sharing techniques as well as the simplicity of the RL interface are the main contributors for the significant reduction of area overhead. Overall, RL has an area saving of 89.6%, 86.1%, 77.9% and 69.9% compared with EVC, Mesh, Bufferless[3] and IMR, respectively.

The wiring area is not included as wires are spread throughout the metal layers and cannot be compared directly. We do acknowledge that IMR and RL use more wiring resources than other designs. RL uses a small percentage of middle metal layers for wires and, as a result, more repeaters are needed. The total area for all the link repeaters is 0.127mm^2 which is 4.3% of the mesh router area. However, as middle layers are above the logic area, RL is unlikely to increase the chip size.

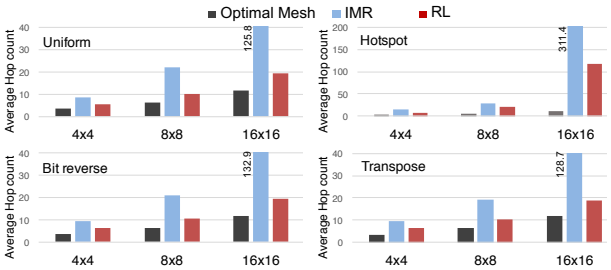

Figure 15: Average hop count for synthetic workloads.

8. DISCUSSION

8.1 Scalability and Regularity

[2]Our CAD tools limit P&R for processing cores.

[3]In addition to area reduction, RL also has 2.8X higher throughput (under UR) and 64.3% lower latency than bufferless NoC.

Table 3: Average overlapping and loops/rings in RL/IMR

Network	Overlap cap	Avg overlap(%) of links	Max loops/ rings in node	Avg loops/ rings(%) in node	Longest loop/ring
RL 4x4	4	3.33 (83.3%)	6	5 (62%)	12
IMR 4x4	4	2.33 (58.3%)	4	3.5 (43%)	14
RL 8x8	8	6 (75%)	14	10.5 (65%)	28
IMR 8x8	8	4.71 (58.9%)	10	8.2 (54%)	48
RL 16x16	16	11.33 (70.8%)	30	21.2 (66%)	60
IMR 16x16	16	8.13 (50.8%)	18	15.2 (47%)	240

Figures 11 and 12 already showed the advantage of RL in terms of latency and throughput for large networks. Figure 15 further compares the average hop count (zero-load hop count) of RL, IMR, and optimal Mesh. As can be seen, IMR has very high average hop count because of its lengthy rings. In contrast, the average hop count of RL is only slightly higher than optimal Mesh. Note that RL achieves this low hop count without having the switch capability of conventional routers.

Routerless NoC is not as irregular as it appears in the figures. In our actual design and evaluation, all the RL interfaces use the same design (some ports are left unused if no loops are connected), so the main irregularity is the way that links form loops. One way to quantify the degree of link irregularity is how many different possible lengths of links, which is $n-1$ for $n \times n$ NoC. This degree is similar to that of Flattened Butterfly [25] and MECS [18].

8.2 Average Overlapping

We discussed before that as long as the overlapping cap is met, it is beneficial to approach this cap for as many neighboring node pairs as possible to increase resource utilization and improve performance. Table 3 presents this statistics for RL and IMR. It can be seen that the average overlapping between adjacent nodes in RL is at least 20% more than that of IMR. Also, the longest loop in RL is always shorter than the longest ring in IMR, and the difference increases as the NoC gets bigger. Shorter loops reduce average hop count and offer a lower latency. For example, in 16×16 the longest loop in RL is of 60 nodes while in IMR it is of 240 nodes.

8.3 Impact on Latency distribution

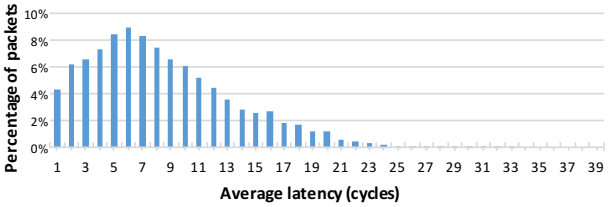

Figure 16: Latency distribution of benchmarks for RL 8×8 NoC.

The extension buffer technique and the reduced ejection link technique save buffer resources at the risk of increasing packet latency. Figure 16 shows distribution of average packet latency, averaged over different benchmarks. The RL interface is configured the same as previous sections with one EXB and two ejectors. The take away message from the figure is that the two techniques has minimal impact on latency under tight resource allocation. For example, the average packet latency is only 8.3 cycles for RL, and only 0.71% of the packets having latency larger than 20 cycles, with the

largest being 39 cycles. The tail in the latency distribution is thin and short.

8.4 RL for n x m Chip

The RL design can be easily extended to any $n \times m$ network sizes. The RL interface design and functionalities remain unchanged. The RLrec algorithm needs to be modified slightly. With rectangular shapes instead of squares, N_L and N_H are not sufficient to denote the four corners of a layer. Two more variables are needed to specify the corners of a layer correctly. For Instance, $N_L r$ and $N_H r$ for low and high rows, and $N_L c$ and $N_H c$ for low and high columns. Once a layer is correctly specified, the four groups of loops can be generated in similar fashion. The rotation step is skipped as this is not possible for rectangular networks, but the reversing direction step remains. The overlapping calculation needs to reflect the orientation of the rectangular loops as well.

9. CONCLUSION

Current and future many-core processors demand highly efficient on-chip networks to connect hundreds or even thousands of processing cores. In this paper, we analyze on-chip wiring resources in detail, and propose a novel routerless NoC design to remove the costly routers in conventional NoCs while still achieving scalable performance. We also propose an efficient interface hardware implementation, and evaluate the proposed scheme extensively. Simulation results show that the proposed routerless NoC design offers significant advantage in latency, throughput, power and area, compared with other designs. These results demonstrate the viability and potential benefits of the routerless approach, and also call for future works that continue to improve various aspects of routerless NoCs such as performance, reliability, and power efficiency.

Acknowledgments

We sincerely thank the anonymous reviewers for their helpful comments and suggestions. We appreciate the authors of IMR [28] for sharing the source code of generating IMR. We also thank Timothy M. Pinkston for providing valuable feedback to the work. This research was supported, in part, by the National Science Foundation (NSF) grants #1619456, #1566637, #1423656, #1619472 and #1321131.

10. REFERENCES

[1] http://ark.intel.com/products/95830/intel-xeon-phi-processor-7290-16gb-1_50-ghz-72-core/.
[2] https://oeis.org/A140517.
[3] http://wccftech.com/intel-sc15-knights-landing-14nm-wafer-specification/.
[4] T. W. Ainsworth and T. M. Pinkston, "On characterizing performance of the cell broadband engine element interconnect bus," in *International Symposium on Networks-on-Chip (NOCS)*, 2007.
[5] R. Arunachalam, E. Acar, and S. R. Nassif, "Optimal shielding/spacing metrics for low power design," in *IEEE Annual Symposium on VLSI*, 2003.
[6] M. Badr and N. E. Jerger, "Synfull: synthetic traffic models capturing cache coherent behaviour," in *ISCA*, 2014.
[7] L. A. Barroso and M. Dubois, "The performance of cache-coherent ring-based multiprocessors," in *ISCA*, 1993.
[8] ——, "Cache coherence on a slotted ring." in *ICPP*, 1991.
[9] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas, "A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *Symposium on VLSI Circuits*, 2016.
[10] L. Chen and T. M. Pinkston, "Nord: Node-router decoupling for effective power-gating of on-chip routers," in *MICRO*, 2012.
[11] B. Christian, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.
[12] I. Cutress, "Supercomputing 15: Intel's knights landing xeon phi silicon on display," November 2015.
[13] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *DAC*, 2001.
[14] G. S. Delp, D. J. Farber, R. G. Minnich, J. M. Smith, and M. C. Tam, "Memory as a network abstraction," *IEEE Network*, vol. 5, no. 4, 1991.
[15] C. Fallin, C. Craik, and O. Mutlu, "Chipper: A low-complexity bufferless deflection router," in *HPCA*, 2011.
[16] C. Fallin, X. Yu, G. Nazario, and O. Mutlu, "A high-performance hierarchical ring on-chip interconnect with low-cost routers," 2011.
[17] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger, "Implementation and evaluation of on-chip network architectures," in *International Conference on Computer Design*. IEEE, 2006.
[18] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *HPCA*, 2009.
[19] D. Harris and N. Weste, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson/Addison-Wesley, 2005.
[20] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, 2007.
[21] J. Howard, S. Dighe, Y. Hoskote, S. Vangal *et al.*, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *IEEE Journal of Solid-State Circuits*, 2011.
[22] C.-H. Jan, U. Bhattacharya, R. Brain, S.-J. Choi, G. Curello, G. Gupta, W. Hafez, M. Jang, M. Kang, K. Komeyli *et al.*, "A 22nm soc platform technology featuring 3-d tri-gate and high-k/metal gate, optimized for ultra low power, high performance and high density soc applications," in *Electron Devices Meeting (IEDM)*. IEEE, 2012.
[23] N. E. Jerger, L. S. Peh, and M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *ISCA*, 2008.
[24] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *ISPASS*. IEEE, 2013.
[25] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *ISCA*, 2007.
[26] A. K. Kodi, A. Sarathy, and A. Louri, "ideal: Inter-router dual-function energy and area-efficient links for network-on-chip (noc) architectures," in *ISCA*, 2008.
[27] J. Liu, L. R. Zheng, D. Pamunuwa, and H. Tenhunen, "A global wire planning scheme for network-on-chip," in *International Symposium on Circuits and Systems (ISCAS)*, 2003.
[28] S. Liu, T. Chen, L. Li, X. Feng, Z. Xu, H. Chen, F. Chong, and Y. Chen, "Imr: High-performance low-cost multi-ring nocs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, 2016.
[29] NanGate, Inc. Nangate freePDK15 open cell library. [Online]. Available: http://www.nangate.com
[30] S. Natarajan, M. Agostinelli, S. Akbar, M. Bost, A. Bowonder, V. Chikarmane, S. Chouksey, A. Dasgupta, K. Fischer, Q. Fu *et al.*, "A 14nm logic technology featuring 2 nd-generation finfet, air-gapped interconnects, self-aligned double patterning and a 0.0588 μm 2 sram cell size," in *IEEE International Electron Devices Meeting*, 2014.
[31] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "Vichar: A dynamic virtual channel regulator for network-on-chip routers," in *MICRO*, 2006.
[32] D. Pamunuwa, J. Oberg, L. R. Zheng, M. Millberg, A. Jantsch, and H. Tenhunen, "Layout, performance and power trade-offs in mesh-based network-on-chip architectures," in *International Conference on Very Large Scale Integration*, 2003.
[33] M. K. Papamichael and J. C. Hoe, "The connect network-on-chip generator," *Computer*, vol. 48, no. 12, 2015.
[34] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian, "Towards scalable, energy-efficient, bus-based on-chip networks," in *HPCA*, 2010.
[35] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob *et al.*, "An 80-tile 1.28 tflops network-on-chip in 65nm cmos," in *ISSCC*, 2007.
[36] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, Eds., *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann Publishers Inc., 2009.
[37] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, 2007.
[38] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, "The potential of the cell processor for scientific computing," in *Computing frontiers*. ACM, 2006.
[39] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *ISCA*, 1995.