Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs

YASHAR ASGARIEH and BILL LIN, University of California, San Diego, California

SMART-based NoC designs achieve ultra-low latencies by enabling flits to traverse multiple hops within a single clock cycle. Notwithstanding the clear performance benefits, SMART-based NoCs suffer from several shortcomings: each router must arbitrate among a quadratic number of requests, which leads to high costs; each router independently makes its own arbitration decisions, which leads to a problem called false negatives that causes throughput loss. In this article, we propose a new SMART-based NoC design called SHARP that overcomes these shortcomings. Our evaluation demonstrates that SHARP increases throughput by up to 19% and average link utilization by up to 24% by avoiding false negatives. By avoiding quadratic arbitration, our evaluation further demonstrates that SHARP reduces the wiring and area overhead significantly.

CCS Concepts: • **Computer systems organization** → **Interconnection architectures**; *Multicore architectures*;

Additional Key Words and Phrases: Networks-on-chip (NoC), single-cycle multi-hop asynchronous traversal (SMART)

ACM Reference format:

Yashar Asgarieh and Bill Lin. 2019. Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs. *ACM Trans. Des. Autom. Electron. Syst.* 24, 6, Article 64 (October 2019), 25 pages.

<https://doi.org/10.1145/3356235>

1 INTRODUCTION

Central to the performance of multi-core processors is the design of the interconnection fabric that enables communications among cores. For scalability, a widely used solution is a Network-on-Chip (NoC) [3, 6, 18, 22, 23]. Communications between cores are achieved by routing messages hop-by-hop from the source to the destination. In particular, NoCs based on a two-dimensional mesh have been widely studied. Although mesh networks are scalable in size, their effectiveness at non-local communications quickly diminishes with increasing number of on-chip cores due to long on-chip latencies, which degrades performance and limits the flexible usage of on-chip resources. Delays due to router pipelines, queuing, and serialization all contribute towards a much longer on-chip latency than an ideal point-to-point interconnect.

Authors' addresses: Y. Asgarieh and B. Lin, University of California, San Diego, 9500 Gilman Dr, La Jolla, CA, 92093; emails: yasgarie@cs.ucsd.edu, billin@eng.ucsd.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Association for Computing Machinery.

1084-4309/2019/10-ART64 \$15.00

<https://doi.org/10.1145/3356235>

Recently, an NoC design called single-cycle multi-hop asynchronous repeated traversal (SMART) [10] has been proposed that enables a flit to traverse many router hops within a single clock cycle, potentially from the source all the way to the destination. Such a multi-hop traversal is referred to as a SMART-hop, and it is made possible by using efficient router-bypass mechanisms and properly engineered wires with asynchronous repeaters. In particular, flits can bypass the pipelines of intermediate routers entirely, resulting in ultra-low latency performance. To enable the dynamic setup of SMART-hops, source nodes broadcast SMART-hop set-up requests (SSRs) via dedicated multi-drop links that are connected to all intermediate routers within some maximum hop distance (HPC_{max}) away. Each router must in turn arbitrate among *all* SSRs within HPC_{max} hops away that are connected to it in which the router potentially serves as either a source, intermediate, or destination router for some SMART-hop. This is to ensure mutual exclusion among competing SSRs that share common input or output links. Finally, router data-paths must be configured accordingly to enable multi-hop traversal.

Although SMART offers notable performance advantages, it also suffers from several shortcomings. First, since each router must consider all SSRs that are within HPC_{max} away, it must arbitrate among up to $HPC_{max} (2HPC_{max} - 1)$ SSRs at each input port. This *quadratic complexity* makes the allocator structure quite expensive in both area and power. Second, all routers *independently* make their own arbitration decisions in parallel, without knowledge of decisions made by upstream routers. This leads to a problem called *false negatives* that causes throughput loss. Third, for the arbitration of competing SSRs and the configuration of data-paths for multi-hop traversals, a large number of interrouter signals must be provided with each dedicated SSR. However, the combined width of all these interrouter signals incurs large wiring overheads. This last shortcoming has been tackled by a recent design called an SSR-Net [5] in which an auxiliary network is introduced for routing most of the interrouter signals in an SSR. This approach significantly reduces the number of wires that must be distributed throughout the chip for broadcasting the SSRs. However, SSR-Net still relies on the quadratic arbitration approach of SMART that independently makes arbitration decisions at each router, without knowledge of arbitration decisions made by upstream routers. As such, the quadratic complexity and false negative concerns remain. Further, each router must still broadcast SSRs via dedicated multi-drop links that are connected to all intermediate routers within HPC_{max} hops away, albeit with a significantly reduced number of wires per SSR.

In this article, we propose a new SMART-based NoC design called SHARP, short for *Smart-Hop Arbitration Request Propagation*. It is based on a *propagation-based SSR arbitration* mechanism that eliminates the quadratic arbitration and false negative shortcomings by only considering the winning SSRs from the previous hop routers, which also remedies the third shortcoming by only propagating the interrouter signals associated with the winning SSR. We further reduce the wiring complexity by only requiring routers to send SSRs to their intermediate neighbors with short wires. Our evaluation demonstrates that SHARP increases throughput by up to 19% and average link utilization by up to 24% by avoiding false negatives. By avoiding quadratic arbitration complexity, our evaluation further demonstrates that SHARP reduces the wiring overhead by up to 61× and 4.3× and the SSR arbitration logic area overhead by up to 31× and 7.8× in comparison to SMART and SSR-Net, respectively.

The rest of this article is organized as follows: Section 2 provides background information. Section 3 and Section 4 present an analysis of the shortcomings of SMART and SSR-Net. Section 5 describes our proposed SHARP design and how it overcomes the aforementioned shortcomings. Section 6 evaluates the impact of the proposed design on performance, area, and energy. Section 7 discusses related work, and Section 8 concludes the article.

Fig. 1. SMART router microarchitecture.

2 BACKGROUND

2.1 Conventional NoCs

In an NoC, packets are divided into flits. Each flit is classified as a head, body, or tail flit, depending on its location within a packet. The head flit is the first flit in a packet and carries destination information. It is followed by one or more body flits and a tail flit. For a virtual channel router, a head flit goes through the following steps at every hop:

- *Buffer Write (BW)*: Writes incoming flits into their VC buffer.
- *Route Computation (RC)*: Selects the departing output port based on the destination.
- *VC Allocation (VA)*: Allocates virtual channel.
- *Switch Allocation (SA)*: Sets switch configuration.
- *Switch Traversal (ST)*: Traverses the switch.
- *Link Traversal (LT)*: Traverses the link to the next router.

Body and tail flits follow the same steps except for route computation and VC allocation since they follow the path as their head flit. Traditionally, the above steps are mapped to a 5-stage router pipeline. Different techniques have been proposed to reduce the number of pipeline stages and per-hop latency [11–14, 16–18].

2.2 SMART NoC

This section describes the key elements of SMART and how it operates. Figure 1 depicts the microarchitecture of a 5-port SMART router for a mesh network. For simplicity, only the *Core_{in}* (C_{in}), *West_{in}* (W_{in}), and *East_{out}* (E_{out}) ports are shown in details.¹ All other input ports are identical to W_{in} , and all other output ports are identical to E_{out} .

In a SMART NoC, conventional clocked link drivers are replaced with asynchronous repeaters at every hop, allowing a flit to traverse multiple hops in a single clock cycle. To enable a single-cycle multi-hop path, SMART adds an alternative data-path to each router to allow a flit to bypass the entire router pipeline and go directly to the next router. This is depicted in Figure 1 with a bold line going from W_{in} to E_{out} .

¹SMART uses a mesh topology for connecting routers; hence, each router has five input/output ports (*West*, *East*, *North*, *South*, and *Core*).

Fig. 2. Traversal over a SMART-hop path.

To facilitate the configuration of a bypass path, each router has a 2:1 multiplexer (mux) at the input of the crossbar to choose between the local buffered flit and the bypassing flit. On a bypass selection, the input buffer will be disabled to avoid latching the bypassing flit, and the crossbar will be configured to connect the input to the proper output port. To control the input buffer, the bypass 2:1 mux, and the crossbar, each router provides a Buffer Write enable (BW_{ena}) signal, a Bypass Mux select (BM_{sel}) signal, and a Crossbar select (XB_{sel}) signal, respectively. The crossbar output is connected to the next router via a repeater that is sized to drive the link as well as the two muxes (2:1 bypass and 4:1 crossbar) at the next router.

An example of a multi-hop traversal (called a SMART-hop) is illustrated in Figure 2: a flit from router R20 travels three hops within a single-cycle, until it is latched at R23. The crossbar at R20 is set to connect C_{in} to E_{out} . The crossbars at R21 and R22 are set to connect W_{in} to E_{out} , with their BM_{sel} set to choose bypass over local. At R23, its BW_{ena} is set to latch the incoming flit. A SMART-hop path can thus be created by appropriately setting BW_{ena} , BM_{sel} , and XB_{sel} at intermediate routers.

To set up SMART-hops, SMART performs switch allocation in two stages: local switch allocation (SA-L) and global switch allocation (SA-G). The SA-L stage is identical to the switch allocation step in a conventional NoC router in which buffered flits at a router arbitrate among themselves to gain access to the output ports. For each winning buffered flit, the source router broadcasts a corresponding Smart-hop Setup Request (SSR). Each of these SSR is sent through dedicated multi-drop wires that are repeated from the source router to all intermediate routers up to HPC_{max} hops away along a chosen path to the destination, where HPC_{max} is the maximum number of hops that a flit can traverse in a single cycle.² The chosen path can either be an XY-path or YX-path.³

Upon receiving the SSRs, the recipient routers conduct a second switch allocation stage called SA-G. In this stage, the recipient router arbitrates among the received SSRs to set up the different parts of a router to operate in bypass (by setting BM_{sel} to bypass) or stop mode (by setting BW_{ena} to latch the incoming flit). This SA-G stage comprises the following two parts:

- **Input Arbitration:** Routers first arbitrate among multiple SSRs to choose the highest priority requests (per input port).
- **Output Arbitration:** A 4-way arbitration is performed after input arbitration to resolve possible conflicts at each output port (e.g., at output E_{out} , winning SSRs from W_{in} , N_{in} , and S_{in} , and possibly a local request from C_{in} are considered to select a winner).

When multiple SSRs from multiple sources arrive at the same time, two priority policies for allocation can be used at the recipient router: (a) *local priority*, which gives higher priority to source routers that are closer, and (b) *bypass priority*, which instead gives higher priority to source routers

²In [10], it was shown that an $HPC_{max} = 8$ is best achievable HPC_{max} for a 2D configuration when energy is taken into consideration.

³Although Figure 2 depicts the SSRs entering a router from $E_{out} \rightarrow W_{in} \rightarrow E_{out}$, the SSRs can come from $N_{in} \rightarrow E_{out}$ or $S_{in} \rightarrow E_{out}$ as well. For example, for router R20 shown in Figure 2, the path can come from N_{in} rather than start from C_{in} in the case of an YX-path.

Fig. 3. SMART pipeline. * indicates head-flit only.

Table 1. Summary of Signal Fields with Each SSR

Primary field used for SSR arbitration		
Signal	Width (in bits)	Description
<i>hop_num</i>	$2 \times \lceil \log_2(1 + HPC_{max}) \rceil$	Specifies the number of hops in a SMART-hop.
Companion SSR fields		
Signal	Width (in bits)	Description
<i>source_id</i>	$2 \times \lceil \log_2(HPC_{max}) \rceil$	Specifies the ID of the source router.
<i>vnet_id</i>	$\lceil \log_2(N_{vnet}) \rceil$	Specifies the ID of the virtual network.
<i>head_flag</i>	1	Indicates that the flit is a head flit.
<i>eject_flag</i>	1	Indicates that the flit should be ejected.
<i>eject_port_id</i>	$\lceil \log_2(N_{port}) \rceil$	Specifies the ID of the ejection port.

that are farther away. When arbitrating among SSRs from routers that are the same distance away, they can be prioritized based on direction. In [10], it was proposed to choose straight-hops > left-hops > right-hops, where straight, left, and right are relative to the input/output port, which we also adopt in our work. As shown in [10], *local priority* performs much better than *bypass priority* under both real and synthetic traffic. Therefore, in the remainder of this article, we only focus on the *local priority* case.

After the switch allocation stages, a flit that won both SA-L and SA-G at its source router will proceed and bypass all the intermediate routers in which its SSR won the corresponding SA-G step. Therefore, a flit needs to spend at least two cycles at the source router before traversing a SMART-hop. The SMART router pipeline is shown in Figure 3. In a nutshell, a SMART-hop traversal is performed as follows:

- At cycle $t - 1$, source routers arbitrate the buffered flits and choose a local candidate for every input/output ports (SA-L).
- At cycle t , SA-L winners broadcast SSRs to routers along their desired path using dedicated wires, and consequently, recipient routers perform SA-G among them by employing a fixed priority scheme and configuring routers accordingly.
- At cycle $t + 1$, SA-G winners traverse the bypass paths that have been set up in a single cycle.

As we shall see in Section 5, our design called SHARP follows the same pipeline, but it uses a propagation-based SSR arbitration scheme instead of the parallel-based SSR arbitration scheme used by SMART, which addresses a number of shortcomings of SMART, as described in Section 3.

2.3 SSR-Net

As observed in [5], for the arbitration of competing SSRs and the configuration of data-paths for multi-hop traversals, a number of signals must be provided with each SSR. These fields are shown in Table 1, where N_{vnet} is the number of virtual networks (e.g., $N_{vnet} = 3$), and N_{port} is the number of router ports (e.g., $N_{port} = 5$). In SMART, all these fields must be broadcast with each dedicated SSR to a quadratic number of routers that are within HPC_{max} hops away, but only *hop_num* is needed for choosing the highest priority request at each input port of a router. Chen and Jha [5] argue

SA-G process split into two stages

Fig. 4. SSR-Net pipeline. * indicates head-flit only.

that this is very expensive in wiring cost because of two reasons: (1) the combined width of all SSR fields is much wider than just the *hop_num* field alone; and (2) all these SSR fields must be included with each of the $HPC_{max}(2HPC_{max} - 1)$ SSRs that each input receives from routers HPC_{max} hops away for SSR priority arbitration.

To avoid broadcasting all the SSR fields shown in Table 1, Chen and Jha [5] propose to split the SA-G process into two pipeline stages: a *pre-SSR* stage and a *post-SSR* stage.⁴ We refer to this design as *SSR-Net* [5]. In particular, in SSR-Net, their pre-SSR stage arbitrates among the $HPC_{max}(2HPC_{max} - 1)$ competing SSRs at each input port. This stage is identical to the parallel-based SSR arbitration scheme in SMART, which only requires the *hop_num* field be broadcast. However, SSR-Net improves significantly in wiring cost over SMART because it does not broadcast the remaining SSR fields with each SSR. Instead, SSR-Net uses an auxiliary network (called an SSR network in [5]) to propagate the remaining *companion* fields that correspond to the winning SSRs from the pre-SSR stage. This propagation of the companion SSR fields happens in a separate *post-SSR* stage, in which the remaining steps of arbitration and bypass configurations are also performed. Therefore, a flit needs to spend at least three cycles at the source router before making a SMART-hop traversal, one more cycle than needed by SMART or SHARP. The SSR-Net router pipeline is shown in Figure 4. As mentioned above, SSR-Net implements SA-G in two stages: SSR priority arbitration is performed in the *pre-SSR* stage using a parallel-based SSR architecture scheme, and the delivery of the companion SSR signals and the bypass configurations are performed in the *post-SSR* stage.

In summary, SSR-Net takes the following steps to setup a SMART-hop:

- At cycle $t - 1$, all source routers perform SA-L to choose a local winner for each input/output port.
- At cycle t , instead of broadcasting all the fields in an SSR, the source routers just broadcast their *hop_num* field to recipient routers HPC_{max} hops away. The recipient routers in turn perform SSR priority arbitration on the received SSRs in their *pre-SSR* stage.
- At cycle $t + 1$, source routers send the remaining companion SSR fields through the auxiliary SSR network in their *post-SSR* stage. Every router then receives the remaining companion SSR fields just for the winning SSRs at each of its inputs and configures itself appropriately for the subsequent single-cycle multi-hop traversal.
- At cycle $t + 2$, flits that won both SA-L and SA-G stages at source routers proceed with their SMART-hop traversals.

3 SHORTCOMINGS OF SMART

In this section, we describe the shortcomings of SMART. We defer to Section 4 to describe the shortcomings of SSR-Net, which are essentially the same as the shortcomings presented in this section.

⁴In [5], the first stage is called a pre-SSR stage, as we also call it in this work, but Chen and Jha [5] refer to the second stage as SA-G, which we find confusing since both stages are part of the overall SA-G process. Therefore, in this work, we instead refer to the second stage of the SA-G process as the post-SSR stage.

Fig. 5. Broadcast SSRs in SMART-2D.

As already mentioned in Section 2.3, SMART incurs large overheads due to the combined width of all SSR fields shown in Table 1 that are broadcast with each SSR. Besides this wiring overhead, which SSR-Net [5] addresses, SMART suffers from several other shortcomings. The first shortcoming is due to the fact that SMART employs a *parallel* SSR arbitration mechanism in which all SSRs that are within HPC_{max} hops away must be considered, which means each router must arbitrate among up to $HPC_{max}(2HPC_{max} - 1)$ SSRs at each input port. This quadratic complexity is expensive both in terms of area and power. The second shortcoming is due to the fact that *all* routers independently make their own allocation decisions in *parallel*, without knowledge of allocation decisions made by upstream routers. This parallel SSR arbitration mechanism leads to a problem called *false negatives* that causes throughput loss. These shortcomings are discussed further below.

3.1 Quadratic Complexity

For the SMART-2D design [10], the SA-G stage considers up to $HPC_{max}(2HPC_{max} - 1)$ SSRs per input port from routers up to HPC_{max} hops away. Figure 5(a) illustrates this for the W_{in} port of router R_{23} with $HPC_{max} = 3$. Note that although the figure shows each source router is connected to R_{23} via a single SSR, sources on the same row may contribute multiple SSRs at W_{in} (i.e., R_{23} may receive up to 3 SSRs from R_{21} and up to 5 SSRs from R_{22}). This is because every path of length $HPC_{max} = 3$ that has its turn position at or after (2, 3) will also go through R_{23} (one dedicated SSR for each). Together, R_{23} may receive up to 15 SSRs at W_{in} from sources within $HPC_{max} = 3$ hops away. A similar analysis can be done for all other directions of R_{23} , and generally for all nodes in a mesh.

Upon receiving the SSRs, the SA-G stage arbitrates among them to choose the highest priority request at each input port. This is depicted in Figure 5(b) for each input in the four mesh directions (West, East, North, South). All these dedicated SSR links together impose an $O(HPC_{max}^2)$ wiring cost on the control plane.

Besides a quadratically increasing wiring cost, the area and energy consumption for the SA-G stage also grow at $O(HPC_{max}^2)$ as well. To understand this, the required logic in SMART for input arbitration, output arbitration, and local configurations are considered, as shown in Figure 6. The

Fig. 6. Implementation of parallel-based SA-G, focusing on W_{in} and E_{out} [10].

input and output signals correspond to the ones shown in the router in Figure 1. Figure 6 is similar to the one shown in [10], which focuses on the logic for SSRs coming in at the input port W_{in} for bypass to the output port E_{out} . We modified the figure from [10] to emphasize two additional details: (1) to determine if an SSR has to terminate or prematurely stop at the current input port, additional *blocking tests* besides checking for a *free* VC at the next-hop are required; (2) besides checking for local requests, the output arbitration step also has to check for competing SSRs from the input ports N_{in} and S_{in} . The functions in Figure 6 are explained as follows:

3.1.1 Input Arbitration. The arbitration process starts by examining up to $HPC_{max}(2HPC_{max} - 1)$ SSRs at each input port to select the highest priority request. A single SSR winner is chosen by comparing *distances* and *directions* of the SSRs relative to the recipient router. In particular, under the *local priority* policy, the *closest* SSR to the recipient router wins the arbitration. In the case of equal-distance SSRs, the authors in [10] proposed to break ties based on directions: straight $>$ left-turn $>$ right-turn. We adopt the same tie-breaker policy in our work.

To determine if a given SSR is an *active* request, the recipient router examines the length information (in hops) that is encoded in the SSR. This length information, referred to as the *hop_num*, indicates the number of hops that the corresponding flit wishes to travel. The recipient router uses this length information, which can be encoded in $\log_2(1 + HPC_{max})$ bits, to determine if the SSR terminates earlier.

Besides prioritizing the SSRs based on distances and directions, input arbitration in SMART also performs *blocking tests* to see if an SSR should be considered for bypass to an output. There are several reasons why an SSR at an input port does not need to be considered for bypass, each of which must be checked during input arbitration.

- Under the *local priority* policy, the arbiter checks to see if there is a local request that won the SA-L stage for the input port, since this local request would have higher priority than requests coming from one or more hops away.
- The arbiter also checks to see if there is a *prematurely* stopped flit from the same source as a given SSR among its buffered flits. If there is, then the SMART-hop path must stop at this input port to ensure the ordering of flits at the destination. In particular, SMART uses a VC allocation table to keep track of the allocations of VCs. The VC allocation is identified by a combination of *source_id* and *vnet_id*. To check if there is a *prematurely* stopped flit among its buffered flits from the same source as the winning SSR, the VC allocation table

Fig. 7. SSR arbitration costs.

is consulted using (*source_id*, *vnet_id*) as the key. If a VC has already been allocated to this source, then the bypassing flit will be stopped at this VC to avoid flit re-ordering.

- In addition, the arbiter checks for the availability of a free VC at the next-hop router for a given SSR. In particular, if the corresponding flit is a head flit, as indicated by *head_flag*, then the immediate downstream router is checked for a free VC to allocate to this head flit (for the corresponding virtual network as specified by *vnet_id*).
- Finally, the arbiter checks to see if the current router is the final destination for a given SSR. To check if the current router is the final destination for a given SSR, *eject_flag* is checked. If this flag indicates ejection, then the corresponding flit will retire directly to the network interface buffer, as specified by *eject_port_id*, bypassing the destination router's pipeline.

Together, input arbitration selects the highest priority SSR that passes the above blocking tests for further arbitration at the corresponding output port.

3.1.2 Output Arbitration and Configuration. Once the winners of the input arbitration part are determined, output arbitration is performed to resolve possible conflicts at each output port (using the same priority policy). Then, the control signals (i.e., BW_{ena} , BM_{sel} , and XB_{sel}) are appropriately set to enable flit traversals in the next cycle.

3.1.3 Complexity. The above analysis for the SA-G logic provides some insights. First, the selection of the highest priority request at each input port is related to the number of SSRs received, which in turn is quadratic with respect to HPC_{max} . However, the complexity of the other arbitration parts is constant, independent of HPC_{max} .

To evaluate the arbitration cost in practice,⁵ we use the Synopsys Design Compiler to synthesize the SA-G logic in a TSMC 45nm technology. Figure 7 depicts the area, power, and latency costs for four HPC_{max} values. In this figure, the table shows the absolute values, and the bar graphs are based on normalized values (normalized to $HPC_{max} = 2$). Indeed, the results clearly show quadratic increases in area and power, and near linear increases in latency, with respect to increasing HPC_{max} . For a large HPC_{max} value, the arbitration logic consumes a significant amount of the area and power budgets of a router and imposes a high latency overhead on the control-path.

3.2 False Negatives and Throughput Loss

As noted in [10], the parallel SSR arbitration scheme used by SMART leads to *false negatives*, a situation in which a router has been setup to bypass or stop an expecting flit, but no flit arrives. This can happen because all routers are *independently* making their own allocation decisions at the same time, *without* knowing the allocation decisions that other upstream routers will make. We illustrate such a false negative scenario in Figure 8. The figure depicts an example in which router R_{20} sends an SSR to R_{22} , and both routers R_{11} and R_{02} send SSRs to R_{23} . In this example,

⁵We refer the reader to [5] for the wiring cost analysis.

Fig. 8. Throughput loss due to false negatives.

SSR_{20} wins output arbitration over SSR_{11} at R_{21} , even though they are equal-distance to R_{21} . This is because straight-hops have higher priority than right-turn-hops in the case of an equal-distance tie breaker [10]. This enables R_{20} to send a flit all the way to its stop router R_{22} , as shown in the solid red line in Figure 8. At R_{22} , SSR_{02} loses output arbitration to SSR_{11} because the straight arrival of SSR_{11} into R_{22} has higher priority than the right-turn arrival of SSR_{02} . This forces the flit from R_{02} to prematurely stop at R_{22} , even though the output link from R_{22} to R_{23} is idle. That is, R_{02} could have sent a flit all the way to its stop router R_{23} , as shown in the partially dashed blue line, but cannot because R_{22} has no way of knowing that SSR_{11} has lost output arbitration at R_{21} .

In the scenario depicted in Figure 8, significant throughput is loss due to false negatives. As we shall see in Section 5, by guaranteeing the avoidance of false negatives, our proposed solution ensures that the bypass paths shown in both solid red lines and partially dashed blue lines in Figure 8 will be used, thus avoiding throughput loss. Further, besides false negatives due to a lack of knowledge regarding which SSRs have already lost SA-G earlier, false negatives can also occur when a higher priority SSR has to prematurely stop at an earlier intermediate router due to one of the blocking conditions described in Section 3.1 (e.g., a flit has to prematurely stop due to an existing buffered flit from the same source at an earlier intermediate router). Our proposed approach also guarantees the avoidance of all these types of false negatives as well.

4 SHORTCOMINGS OF SSR-NET

As discussed in Section 2.3, SSR-Net splits the SA-G process of SMART into two pipeline stages: a *pre-SSR* stage and a *post-SSR* stage [5]. The pre-SSR stage is identical to the parallel SSR priority arbitration step in SMART, as explained in Section 3.1, which only requires the *hop_num* field be broadcast. Like SMART, each router in SSR-Net still *independently* makes its own SSR arbitration decision, without knowledge of arbitration decisions made by upstream routers. As such, SSR-Net suffers from the same shortcomings as SMART, as described in Section 3. Specifically, SSR-Net suffers from the same quadratic complexity and false negative problems that afflict SMART.

5 SHARP NOC

In this section, we describe our proposed design named SHARP, for *Smart-Hop Arbitration Request Propagation*. Instead of arbitrating among $O(HPC_{max}^2)$ SSRs in parallel at each input port, as SMART and SSR-Net⁶ do, SHARP eliminates the quadratic arbitration problem by only considering at each

⁶As already mentioned, the pre-SSR stage of SSR-Net is identical to the parallel-based SSR arbitration scheme in SMART.

input port the *winner* from the corresponding output port at the previous hop router. The previous hop router in turn only has to consider the winners from its previous hop routers that are adjacent to its input ports, and so on. We refer to this arbitration method as *propagation-based* SSR arbitration. As we shall see, besides eliminating the quadratic complexity of parallel-based SSR arbitration used by SMART and SSR-Net, SHARP also guarantees the avoidance of false negatives. That is, SHARP obviates the two shortcomings of SMART and SSR-Net, as outlined in Section 3 and Section 4. In the following, we further explore the properties of parallel SSR arbitration. We then apply the insights into the design of SHARP, and we analyze the delays through the bypass path setups and flit traversals. Finally, we discuss about the delivery of companion SSR signals. In this remainder of this section, we will primarily draw comparisons with SMART when discussing about SHARP, but the same comparisons also apply to SSR-Net as SSR-Net uses the same parallel-based SSR arbitration scheme in SMART.

5.1 Redundancies and Logical Dependencies

Consider again Figure 5(a) where R_{23} receives up to 15 SSRs at its W_{in} port from sources within $HPC_{max} = 3$ hops away. In this example, we observe two properties:

- (1) *Redundancies*: We first observe that the set of SSRs considered at the W_{in} port of R_{23} is a *subset* of SSRs considered at R_{22} , including local requests at R_{22} . In particular, the subset is *exactly* the set of SSRs that are (a) within $(HPC_{max} - 1)$ hops away from R_{22} and (b) go out through the E_{out} port of R_{22} , including the local request that won SA-L at E_{out} of R_{22} . For example, in Figure 5(a), SSRs from R_{22} , R_{21} , R_{12} , R_{32} , R_{20} , R_{11} , R_{31} , R_{02} , and R_{42} (shown in light grey nodes) are considered by the input arbitration step at W_{in} of R_{23} to select a winner, as these SSRs are within $HPC_{max} = 3$ hops away. However, the same set of SSRs, including the local request from R_{22} itself, must also be considered at R_{22} to select a winner at its E_{out} port, as these SSRs are within $(HPC_{max} - 1)$ hops away from R_{22} .

We further observe that in parallel SSR arbitration, the input arbitration step at the W_{in} port of R_{23} will select the *exactly same* SSR as winner as the output arbitration step at the E_{out} port of R_{22} . The same observation can be made for any input port with the output port of the corresponding previous hop router (e.g., the winner at input port N_{in} of R_{22} is the same as the winner of the output port S_{out} of R_{12}). This means that the SSR arbitration step at each input port is *redundant* in the sense that the same arbitration logic and the same arbitration result are already required and available at the corresponding previous hop router. In turn, the output arbitration step at the previous hop router only needs to know about the winners at its input ports, but the arbitration logic and results for these input ports are already required and available at their corresponding previous hop routers as well, and so on. This redundancy imposes significant overhead in power, area and performance as the redundancy is replicated in parts at *every* router in the network. This is necessary because in parallel SSR arbitration, all routers independently make their own allocation decisions.

Alternatively, we can simply eliminate the SSR arbitration step (and the associated quadratic arbitration problem) all together at each input port by *propagating* the winner from the output arbitration step at the previous hop router. However, at each input port, we still have to perform the *blocking tests*, as described in Section 3.1, to determine if the propagated winning SSR needs to terminate or prematurely stop at the current input port.

- (2) *Logical Dependencies*: We observe that the priorities of SSRs are *distance-based*. In particular, under the *local priority* policy, an SSR 1-hop away can only be granted if there is no competing local request. Similarly, an SSR 2-hops away can only be granted if there are

(a) Implementation of propagation-based SA-G.

(b) Propagation of the winning SSRs.

Fig. 9. Implementation of propagation-based SA-G, focusing on W_{in} and E_{out} .

no competing SSRs from 1-hop away. In turn, an SSR k -hops away can only be granted if there are no competing SSRs from $(k - 1)$ -hops away, and so on. Therefore, the arbitration among SSRs that are k -hops away is *logically dependent* upon the arbitration result among SSRs that are $(k - 1)$ -hops away. This logical dependency explains the linearly increasing latencies observed in Figure 7 with respect to HPC_{max} for the critical path through the parallel SSR arbitration logic implementations. As we shall see next, this logical dependency is already captured in our propagation-based arbitration approach in the sense that arbitration decisions made at a node $(k - 1)$ -hops away are logically dependent upon the winners selected at routers k -hops away. The important difference, however, is that the *redundant* logic is eliminated.

5.2 Propagation-Based SSR Arbitration

Figure 9 depicts our *propagation-based* SSR arbitration approach in SHARP for SA-G. Instead of arbitrating among up to $HPC_{max}(2HPC_{max} - 1)$ SSRs in parallel at each input port, as SMART does, SHARP entirely eliminates the quadratic *SSR priority arbitration* function shown in Figure 6 by exploiting the fact that the winning SSR at a given input port is the *same* as the winner from the corresponding output arbitration step at the previous hop router. Therefore, we can simply *propagate* that winner from the previous hop router. This winner is depicted as $SSR_{W_{in}}$ in Figure 9(a) and Figure 9(b) at the input port W_{in} , which comes from the output port E_{out} at the previous hop router. At the input port, we now simply have to perform the *blocking tests* on $SSR_{W_{in}}$, as described

in Section 3.1, to determine if a bypass path should be set up. The output arbitration step is nearly identical to SMART, except that we propagate the winning SSR to the output port E_{out} to the next hop router. Finally, the local configuration step is essentially the same as SMART. The key to SHARP is that the implementation complexity of all three steps, blocking tests, output arbitration, and local configuration, is independent of HPC_{max} and the number of routers in the network.

5.3 Implementation Details

In SMART, each dedicated SSR uniquely identifies the start router and the path (i.e., an XY or YX path) to the recipient router. The *hop_num* field indicates to the recipient router whether the SSR terminates at or before the recipient router, or passes through the recipient router. The uniqueness of each dedicated SSR, together with the *hop_num* field, also enables a recipient router to determine if the requested path makes a turn or not at the current router, and if the turn should be to the left or right.

In SHARP, the wires used to carry *information* about the winning SSR to an input port do not uniquely identify a start router—the wires could carry information about *any* SSR within HPC_{max} hops away from the recipient router to its input port. Instead of a *hop_num* field, SSRs in SHARP are encoded by a *stop_id* field and a *distance* field. The *stop_id* in (x, y) coordinates is the location of the stop router where the SSR wishes to terminate. This information is sufficient for a recipient router to determine whether the SSR terminates at or before the recipient router, or passes through the recipient router (e.g., the recipient router can test if the (x, y) coordinates of the *stop_id* is the same as its own (x, y) coordinates). This information is also sufficient for a recipient router to determine if the requested path makes a turn or not at the current router, and if the turn should be to the left or right (e.g., if the (x, y) coordinates of the *stop_id* and the $(\hat{x}, y)$ coordinates of the recipient router are such that $\hat{x} < x$ and the y -coordinates are the same, then a left turn should be made).

The *distance* field, which gets incremented at each hop, is used by the output arbitration step in SHARP to determine which of the remote SSRs from the input ports at the recipient router has the highest priority. For a tie-breaker, the *direction*-based priority can simply be determined by the arrival input port of the corresponding SSR (i.e., for E_{out} , the priority straight > left-turn > right-turn is equivalent to $SSR_{Win} > SSR_{Nin} > SSR_{Sin}$).

Aside from a different way of implementing the SA-G step, as depicted in Figure 9, and a different way to encode the SSRs, as explained above, SHARP follows essentially the same pipeline structure as SMART, as depicted in Figure 3.

5.4 HPC_{max} Analysis

In this section, we analyze the maximum number of hops (HPC_{max}) that can be traversed within a single cycle using SHARP. To compare with SMART, we analyze delays using the same 45nm technology node used in the analysis for SMART [10]. Using 1mm per hop and a 1ns clock, [10] showed that an HPC_{max} of 13 or more hops could be achieved for repeated wires at 45nm. However, as discussed in Section 2.2 and as shown in Figure 2, the delay at each hop includes not just the repeater and link segment delay, but also the delay through the bypass and crossbar muxes (as set by BM_{sel} and XB_{sel} , respectively). In particular, the delay through the datapath for traversing up to HPC hops is given by Equation (1), where t_{bypass} is the mux delays through the bypass and crossbar muxes, and t_ℓ is the link segment delay per hop.

$$T_{data} = HPC \cdot (t_{bypass} + t_\ell). \quad (1)$$

The added mux delays per hop reduces HPC_{max} to 11 at 1GHz, as explained in [10].

However, as shown in [10], the achievable HPC_{max} is actually limited by the delay through the SMART parallel SSR arbitration control path. The delay through the SMART control path includes the link delays for the farthest SSRs to reach the recipient router, plus the delay through a parallel SA-G stage that must consider up to $HPC_{max}(2HPC_{max} - 1)$ SSRs. This delay for HPC hops is given by Equation (2), where t_ℓ is again the link segment delay per hop for carrying the SSRs, and t_{sa-g}^{smart} is the delay through a parallel SA-G stage.

$$T_{ctrl}^{smart} = HPC \cdot t_\ell + t_{sa-g}^{smart} \quad (2)$$

As shown in [10], $HPC_{max} = 8$ is the optimal achievable HPC_{max} at 1GHz.

For the SHARP NoC design, the achievable HPC_{max} is also limited by the SA-G control path. The delay through the SHARP control path includes the link delay at each hop, plus the SSR arbitration delay at each hop. This delay for HPC hops is given by Equation (3), where t_ℓ is again the link segment delay per hop for propagating the winner SSR from one hop to the next, and t_{sa-g}^{sharp} is the SSR arbitration delay per hop for each propagation-based SA-G stage.

$$T_{ctrl}^{sharp} = HPC \cdot (t_\ell + t_{sa-g}^{sharp}) \quad (3)$$

To derive the achievable HPC_{max} for SHARP, we use the same repeated wire configuration as Krishna et al. [10] to obtain t_ℓ , and we use the Synopsys Design Compiler to synthesize our propagation-based SA-G logic in a TSMC 45nm technology to obtain t_{sa-g}^{sharp} . Our synthesis results show that an $HPC_{max} = 6$ can be achieved. The total contribution from $HPC \cdot t_\ell$ is 414ps, and the total contribution from $HPC \cdot t_{sa-g}^{sharp}$ is 534ps, which together is under 1ns. Although our SSR arbitration logic is considerably simpler than SMART's, we nonetheless have a longer delay through the control path as the delay through our SSR arbiter has to be multiplied HPC times. As we shall see in Section 6, even though we have a smaller HPC_{max} , SHARP actually performs better than SMART for two reasons: (1) SHARP avoids false negatives, which leads to higher throughput; and (2) With increasing traffic, long multi-hop paths become increasingly difficult to set up when the *local priority* policy is used. This is because multi-hop paths often have to prematurely stop to give way to competing local requests. Therefore, a higher HPC_{max} value is not always helpful.

5.5 Avoidance of False Negatives

As discussed in Section 3.2, a shortcoming of SMART is that routers independently make their own allocation decisions without knowledge of allocation decisions made by upstream routers. For example, an SSR may no longer be active either because it lost SSR arbitration earlier or had to stop prematurely due to the failure of a blocking test. This leads to false negatives that cause throughput loss. In contrast, SHARP only propagates an SSR to the next hop only if it passes all blocking tests and *wins* SSR arbitration. Therefore, false negatives cannot occur (or are guaranteed to be avoided). Figure 10 shows the same example used to illustrate the false negative problem in Figure 8 of Section 3.2. In particular, Figure 10 illustrates the avoidance of false negatives in the *local priority* case. Because SSR_{20} wins output arbitration over SSR_{11} at R_{21} , only the winning SSR_{20} gets propagated to R_{22} where it terminates. At R_{22} , SSR_{02} is the only request competing for the output port to R_{23} , and therefore wins by default. That is, SSR_{21} does not prevent SSR_{02} from winning as it does in the case of SMART, which allows R_{02} to send a flit all the way to R_{23} . Therefore, both R_{20} and R_{22} can send their flits all the way to their respective stop routers, as shown in the solid red and blue lines in Figure 10, which again results in higher throughput. As we shall see in Section 6, the avoidance of false negatives is crucial for throughput at higher traffic loads when contentions among competing SSRs are fierce. Also, as we shall see in our evaluations, the avoidance of false negatives is crucial for setting up longer multi-hop paths at higher traffic

Fig. 10. SHARP avoids false negatives.

Table 2. How SSR Control/Data Signals Are Transmitted

Category	Fields	SMART	SSR-Net	SHARP
SSR arbitration	hop_num	*	*	
	stop_id			→
	distance			→
Companion SSRs	source_id	*	→	→
	vnet_id	*	→	→
	head_flag	*	→	→
	eject_flag	*	→	→
	eject_port_id	*	→	→
Data	flit_data	→	→	→
	flit_metadata	→	→	→
	VC_control	→	→	→

*: broadcast

→: propagated

loads that would otherwise be prematurely stopped. For example, in Figure 10, SSR_{02} would have prematurely stopped in SMART at R_{22} .

5.6 Delivery of Companion SSR Signals

Rather than receiving up to $HPC_{max}(2HPC_{max} - 1)$ SSRs in parallel at each input, as SMART and SSR-Net⁷ do, SHARP simply receives one winning SSR that corresponds to the winner from the corresponding output arbitration step at the previous hop router. The propagation of the winning SSRs also includes the propagation of the corresponding companion SSR signals. Thus, like SSR-Net, SHARP avoids the broadcasting of the companion SSR signals from a quadratic number of SSRs from HPC_{max} hops away. SHARP further reduces wiring cost because it also avoids the broadcasting of the control signals that are needed for SSR priority arbitration from a quadratic number of SSRs from HPC_{max} hops away, as SMART and SSR-Net require (i.e., the *hop_num* field). SHARP simply propagates the control signals (i.e., the *stop_id* and *distance* fields) of the winning SSRs from one router to the next. Table 2 summarizes how the different SSR control signals are transmitted in SMART, SSR-Net, and SHARP. For completeness, Table 2 also summarizes data signals (corresponding to the routing of flits) that are propagated from one router to the next.

⁷SSR-Net needs at least the *hop_num* field with each SSR for the pre-SSR stage.

Table 3. System Parameters and Benchmarks

Processor	
Core	Frequency: 1GHz; #Cores: 64; Tile Area: $1\text{mm} \times 1\text{mm}$; Technology: 45nm
Caches	L1: 64 private 4-way, 32KB/core; L2: 64 shared, 8-way, 1MB; Coherence: MOESI (blocking)
NoC	
Router	Frequency: 1GHz; Virtual Networks: 3; VCs/virtual network: 4; Routing: XY/YX
Interconnect	Topology: 8×8 mesh; Link: 1mm; Width (flit): 128-bit
Applications	
Synthetic	uniform, tornado, bit-compliment
Applications	fft, barnes(br), lu_cb(lc), lu_ncb(lnc), radix(rd), bodytrack(bt), cholesky(ck), facesim(fs), blackscholes(bs), swaptions(sw), water_nsquared(wns), radiosity(rs), raytrace(rt)

6 EVALUATION

6.1 Experimental Setup

For our evaluations, we consider an 8×8 mesh network with 64 cores. Each core comprises a processor node, an L1 data/instruction cache, a slice of a shared L2 cache, and a slice of the cache coherency directory. The cores operate on a 1GHz clock. We allocate a 32KB private 4-way L1 cache to each of the 64 cores, and we allocate a 1MB shared 8-way L2 cache. For cache coherence, we employ a MOESI directory protocol [8]. We assume each core occupies a $1\text{mm} \times 1\text{mm}$ area with 1mm SMART-hop links between them. For SMART, SSR-Net, and our SHARP approach, we consider both XY and YX routing paths. Therefore, for SMART and SSR-Net, SSR dedicated links can stretch along both XY and YX directions. For our evaluations, we consider $HPC_{max} = 8$ for SMART and SSR-Net, which is the best achievable HPC_{max} for a 2D configuration [10]. For SHARP, we consider $HPC_{max} = 6$, which is the best achievable HPC_{max} for SHARP under the same system configuration. For synthetic benchmark evaluation, all packets are fixed at 1 flit/packet. For full-system evaluation, two types of packets are considered: control packets are 1 flit/packet, and data packets are 5 flits/packet. A head flit goes through a three- or four-stage pipeline in the SMART and SSR-Net approaches, respectively, as shown in Figures 3 and 4. SHARP follows essentially the same three stage pipeline as SMART, as depicted in Figure 3. For the memory subsystem, we use a shared L2 cache because of its higher injection rate over a private type. These system parameters are summarized in Table 3.

For performance evaluation, we use Garnet [1] for NoC simulation and gem5 [4] for full-system simulation. For area and energy evaluations, we follow the same methodology as used in [5] by extending the NoC area and power estimation tool DSENT [21] to model SMART, SSR-Net, and SHARP. The area and energy results are calculated using DSENT with a 45nm technology node library. The networks are evaluated by running both synthetic and real application benchmarks. For synthetic traffic, we use three traffic models: Uniform, Tornado, and Bitcomp (bit-complement). Under Uniform traffic, each core will generate traffic to a destination chosen at uniform random. Under Tornado traffic, a core at location (x, y) will generate traffic to a destination at location (dx, y) , where $dx = (x + 3) \bmod k$, where k is the radix of the mesh network (e.g., $k = 8$). Under Bitcomp traffic, a core at location (x, y) will generate traffic to a destination at location (dx, dy) , where dx and dy are bitwise complements of the binary encodings for x and y , respectively (e.g., if $x = 000$, then $dx = 111$). For real-case scenarios, we use selected applications from the PARSEC and SPLASH-2 benchmark suites and ran them through the gem5 full-system simulator. These benchmarks are also summarized in Table 3.

6.2 Performance Comparisons

We first compare the performance of SMART, SSR-Net, and SHARP for average flit latency and link utilization under synthetic traffic patterns. As a baseline, we also compare our proposed design with a conventional hop-by-hop routing approach. In particular, we assume a 1-cycle router with a single-cycle link traversal as this is the state-of-the-art in conventional hop-by-hop routing [19]. We refer to this approach as “Baseline.” The results are presented in Figures 11 and 12. For the latency results shown in Figures 11(a) to 11(c), we see that all three SMART NoC architectures, including SHARP, achieve substantially lower packet latencies than conventional NoC approach (“Baseline”) under low-load scenarios. At near-zero load, the average packet latency of a single-cycle router implementation for a conventional NoC is 3.15X, 2.75X, and 2.47X higher than SHARP for tornado traffic, uniform traffic, and bit-complement traffic, respectively. The conventional NoC approach incurs longer latencies is because it has to perform hop-by-hop routing, so many cycles are still required when the destination is many hops away, where as SMART NoC architectures (including SHARP) can traverse multiple hops in a single cycle.

When comparing the SMART NoC architectures, the latencies at low-loads are comparable. We see that SHARP has slightly higher latencies for low injection rates, but much better latencies for moderate to high injection rates, in comparison to SMART and SSR-Net. These observations show that the improvement in the network capacity is negligible for the Tornado, but is quite substantial for the Uniform and Bitcomp traffic patterns. The results show 19% and 10% improvement in throughput over SMART and SSR-Net for Uniform and Bitcomp, respectively.

Figures 12(a) to 12(c) show the link utilization results, where the average link utilization corresponds to the average number of network links being used per cycle (i.e., with a flit traversing it). The results in Figures 12(a) to 12(c) show that the link utilization is almost identical for low injection rates, but SHARP significantly outperforms SMART and SSR-Net in link utilization for moderate to high injection rates. These observations show that the improvement in average link utilization is negligible for the Tornado, but is quite substantial for the Uniform and Bitcomp traffic patterns. The results show 24% and 11% improvement in average link utilization over SMART and SSR-Net for Uniform and Bitcomp, respectively. The network latency and link utilization results, which are correlated, clearly demonstrate the inefficiencies that SMART and SSR-Net face due to the *false negatives* problem described in Section 3. It is worth noting that both SMART and SSR-Net have similar network latencies and link utilizations because both employ the same parallel SSR arbitration mechanism. The difference in network latencies between SMART vs. SSR-Net is due to the extra pipeline stage required by the SSR-Net approach. For SHARP, SSR arbitration is performed by propagating the highest priority SSR to the next router. Therefore, the avoidance of false negatives is guaranteed, as explained in Section 5.

6.3 Wiring and Area Comparisons

In this section, we compare SHARP to SMART and SSR-Net in terms of wiring and area costs. We first compare the wiring costs of SHARP vs. SMART and SSR-Net. For all three designs, Table 1 specifies the *data* signals associated with the flit being forwarded and the *control* signals needed for SSR arbitration. For SSR-Net, we also implemented a *coarse* version with a single bit-width pre-SSR, as proposed in [5] for reducing wiring costs⁸ to represent the best that the other techniques can achieve in terms of wiring cost. We label this coarse version of SSR-Net as SSR-Net[©].

In Figure 13(a), we compare the wiring costs for SMART, SSR-Net, and SHARP with respect to $HPC_{max} = 1, 2, 4, 6, \text{ and } 8$. The wiring costs are normalized to the cost of the data signals to

⁸Using coarse pre-SSRs significantly exacerbates the *false negatives* problem and increases dynamic energy consumption, especially for large HPC_{max} values.

(a) Tornado Traffic.

(b) Uniform Traffic.

(c) Bitcomp Traffic.

Fig. 11. Average network latency. SHARP achieves better network latencies due to the guaranteed avoidance of false negatives.

(a) Tornado Traffic.

(b) Uniform Traffic.

(c) Bitcomp Traffic.

Fig. 12. Average link utilization. SHARP achieves higher link utilizations due to the guaranteed avoidance of false negatives.

(a) Wiring cost comparisons.

(b) SSR control logic area comparisons.

Fig. 13. SSR arbitration wiring and logic area costs.

highlight the overhead of the control signals required for each of the three approaches. That is, we divide the wiring cost of the control signals by the wiring cost of the data signals, where the wiring costs for the data signals are the same for all three approaches, but the wiring costs for the control signals are considerably higher for SMART and SSR-Net. Recall that in SMART and SSR-Net, the parallel SSR arbitration approach needs to arbitrate among up to $HPC_{max}(2HPC_{max} - 1)$ SSRs per input port at each router from routers up to HPC_{max} hops away. Therefore, we expect to see a *quadratic* increase in wiring costs with increasing HPC_{max} . This quadratic increase can indeed be observed in Figure 13(a) for SMART, SSR-Net, and SSR-Net©. As explained in Section 2.3, SSR-Net and SSR-Net© improve over SMART in wiring cost due to the use of an SSR auxiliary network to propagate companion SSR signals instead of each router broadcasting all companion SSR signals in parallel to all routers HPC_{max} away. However, we can observe still a quadratic increase in wiring cost for SSR-Net and SSR-Net© due to the fact that each router still needs to broadcast in parallel the main SSR control signal (*hop_num*) to all routers that are HPC_{max} hops away. That is, each input port at each router still receives up to $HPC_{max}(2HPC_{max} - 1)$ SSRs to perform the parallel SSR arbitration step. Recall that in SHARP, SSR arbitration is performed by propagating the highest priority SSR to the next router. Therefore, each router just needs to consider one SSR at each input port. Nonetheless, we can observe that the wiring cost for SHARP does increase, albeit slowly, with increasing HPC_{max} . This is due to the fact that the width of some SSR control signals (e.g.,

hop_num) are increasing logarithmically with respect to HPC_{max} (see Table 1). Indeed, as shown in Figure 13(a), the wiring cost for SHARP increases very slowly with increasing HPC_{max} . For $HPC_{max} = 6$, the wiring overheads for the SSR control signals of SSR-Net and SMART are $2.63\text{--}8.67\times$ the wiring cost of the data signals, respectively, which is to be expected due to the quadratic increase in wiring costs. In the optimized coarse version of SSR-Net, labeled as SSR-Net $\odot$, the wiring overhead for the SSR control signals is a fraction of the wiring cost of the data signals by $0.61\times$. However, in the case of SHARP, the wiring overhead for the SSR control signals is even smaller, only $0.142\times$ the wiring cost of the data signals. This corresponds to a reduction by $61\times$ and $4.3\times$ in wiring cost in comparison to SMART and the optimized coarse version SSR-Net $\odot$, respectively.

We next compare the area costs of the SSR control logic for SHARP vs. SMART and SSR-Net. In Figure 13(b), we compare the area costs of the SSR control logic for all three approaches. We again compare the area costs for SHARP, SSR-Net, and SHARP with respect to $HPC_{max} = 1, 2, 4, 6$, and 8 . The area costs shown in Figure 13(b) are normalized to the SSR control logic cost with $HPC_{max} = 1$. As shown in Figure 13(b), the SSR control logic cost is essentially the same for $HPC_{max} = 1$ for all three approaches since all three approaches consider exactly the same number of SSRs, one SSR per input port. Just as we can observe for the wiring costs, the area costs for the SSR control logic for SMART and SSR-Net are also increasing *quadratically* with respect to HPC_{max} since both SMART and SSR-Net have to consider a quadratically increasing number of SSRs with respect to HPC_{max} . On the other hand, with SHARP, we can observe that the area cost for the SSR control logic increases slowly with respect to HPC_{max} . This is again due to the fact that SHARP performs SSR arbitration by propagating the highest priority SSR to the next router. Therefore, each router just needs to consider one SSR at each input port. The increase in area observed in Figure 13(b) is again due to the fact that the width of some SSR control signals (e.g., hop_num) are increasing logarithmically with respect to HPC_{max} (see Table 1), which leads to an increase in the SSR control logic area. For $HPC_{max} = 6$, the area overheads of SSR-Net and SMART are $26\text{--}102\times$ the baseline area for the $HPC_{max} = 1$ case, which again is to be expected due to the quadratic increase in the number of SSRs that have to be considered. On the other hand, the area overhead of SHARP is only $3.33\times$ the baseline area for the $HPC_{max} = 1$ case. This corresponds to a reduction by $31\times$ and $7.8\times$ in area overhead in comparison to SMART and SSR-Net, respectively.

6.4 Energy Comparisons

In this section, we compare SHARP to SMART and SSR-Net in terms of energy consumption. In particular, we break down the average dynamic energy consumed per flit. For this analysis, we again assume an 8×8 network with an $HPC_{max} = 8$ for SMART and SSR-Net and an $HPC_{max} = 6$ for SHARP, which are the best achievable values for the proposed system configuration. We use the Uniform traffic pattern to determine the dynamic energy consumption. The results are shown in Figure 14 for three injection rates, low, moderate, and high, which are set to $0.04, 0.14$, and 0.24 flit/node/cycle, respectively. The results for each injection rate are normalized to the energy consumption of SMART at that injection rate. That is, we are using SMART as the baseline for comparison.

When the injection rate is low, we observe that SSR-Net and SHARP consume 7% less energy than SMART because of the energy reduction in the SSR arbitration logic. When the injection rate is moderate, the contribution from the arbitration logic to the total energy consumption increases. In particular, the energy per flit reduces by 4% and 8% for SSR-Net and SHARP, respectively. We can infer from these results that for a moderate injection rate, the increase in energy consumption due to buffering is almost offset by the reduction in energy consumption for the SSR arbitration logic. When the injection rate is high, the networks are still only moderately utilized. At this injection

Fig. 14. Energy consumption comparisons.

rate, SMART and SSR-Net both suffer from high rates of false negatives, which means that the SSR arbitration logic at most nodes are active and consume power. Therefore, the arbitration logic has a larger share of the total dynamic energy per flit. For SHARP configurations, a negligible part of the total energy comes from the arbitration logic since SHARP guarantees the avoidance of false negatives, and therefore it has a significantly simpler arbiter. To summarize, the arbitration logic in SMART and SSR-Net consumes more energy, despite being increasingly inefficient at utilizing links effectively at higher injection rates.

6.5 Full-System Evaluation

For full-system evaluation, we evaluate our design by running selected applications from the PARSEC and SPLASH-2 benchmark suites. We evaluate these benchmarks on a 64-core (8×8) system with 64 threads enabled, again using $HPC_{max} = 8$ for SMART and SSR-Net and an $HPC_{max} = 6$ for SHARP. Figure 15 shows the average flit latency comparisons for the three approaches. All results shown in Figure 15 are normalized to SMART as the baseline.

For all benchmarks evaluated under full-system simulation, the injection rates fall in the lower end of the injection rates in the graphs for synthetic traffic. At these low injection rates, the higher HPC_{max} that SMART can achieve is beneficial in achieving a lower average flit latency since false negatives are less likely in this regime. This is why SHARP has a higher average flit latency than SMART. However, as observed in Section 6.2, the benefit of a higher HPC_{max} for SMART and SSR-Net quickly diminishes with increasing injection rates due to the throughput loss caused by increasing probabilities of false negatives. The reason why SSR-Net has a higher average flit latency than SMART, despite having the same HPC_{max} , is due to the extra pipeline stage in SSR-Net.

Overall, the differences in latencies between SMART, SSR-Net, and SHARP for these real-application benchmarks are small. However, SHARP offers significant benefits over SMART and SSR-Net in that SHARP achieves higher throughput (as shown in Figure 11 and Figure 12)

Fig. 15. Full-system evaluation of real application benchmarks.

by avoiding false negatives, and SHARP significantly reduces the wiring and area overhead (as shown in Figure 13 and Figure 14) by avoiding quadratic arbitration.

7 RELATED WORK

In this section, we discuss related work for reducing packet latencies in NoCs. As discussed in Section 2, a number of different techniques have been proposed to reduce the number of pipeline stages and per-hop latency, including techniques for simplified VC selection [11], speculative switch arbitration [14, 16, 18], and lookahead computation [12, 13, 17].

Topology solutions have also been proposed to reduce average hop counts by reducing the network diameter and connecting distant routers with dedicated links. Examples of this approach include the Flattened Butterfly [9] and MECS [7]. However, the reduced network diameter comes with a price: high-radix routers with many more ports must be used, which makes each router considerably more complex (buffers are needed at more ports, the crossbar grows quadratically with the number of ports, route computation and switch arbitration become more complicated, etc.). To achieve similar area and power as a typical 5-port router used in a conventional 2D-mesh, the channel bandwidth is often reduced proportionally (by reducing the channel width). Another reason for reducing the channel width is because NoCs with high-radix routers typically have many more channels crossing the network bisection due to many global wires that must span across the chip. As shown in [10], a Flattened Butterfly with the same bisection bandwidth as a SMART NoC actually achieves much lower throughput and higher latency due to heavy serialization delay.⁹ Furthermore, while topology solutions like the Flattened Butterfly work for certain classes of traffic, the added serialization delays result in higher latencies for local traffic (such as nearest neighbor).

Another approach for reducing the network diameter is by concentrating multiple cores to a router. CMesh [2] is an example of this approach where cores are concentrated 4-to-1 in a 2D mesh. While the average number of hops is cut in half, multiple hops are still required, with intermediate queuing delays. In [15], network concentration is dynamically configured, depending on the traffic mix. SHARP and SMART are complementary to these network concentration approaches in that the SHARP and SMART designs can be used to dynamically configure single-cycle multi-hop paths across a concentrated mesh.

⁹Packets must be serialized to a reduced channel width. As shown in Section 6.2, in comparison to SMART, SHARP achieves higher throughput as well as lower latencies at moderate to high injection rates. Therefore, SHARP will also significantly outperform a Flattened Butterfly with the same bisection bandwidth.

At the extreme end of concentration, Swizzle-Switch [20] proposes to use a single switch to connect all cores on a chip. In [20], it was shown that a highly customized Swizzle-Switch at the layout-level with specialized circuits can connect up to 64 cores. However, the scalability of this approach remains to be shown since the switch grows quadratically with the number of cores.

8 CONCLUSION

We presented a new NoC design called SHARP that avoids several inherent shortcomings of an earlier design called SMART, which suffers from quadratic complexity and throughput loss due to false negatives. It is based on a propagation-based SSR arbitration scheme that only considers the winning SSRs from the previous hops. SHARP also reduces wiring by only propagating the interrouter signals associated with the winning SSR.

REFERENCES

- [1] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *ISPASS*. IEEE Computer Society, 33–42. <http://dblp.uni-trier.de/db/conf/ispass/ispass2009.html#AgarwalKPJ09>.
- [2] James Balfour and William J. Dally. 2006. Design tradeoffs for tiled CMP on-chip networks. In *ACM International Conference on Supercomputing (ICS)*. 187–198.
- [3] Luca Benini and Giovanni De Micheli. 2002. Networks on chips: A new SoC paradigm. *IEEE Computer* 35, 1 (2002), 70–78.
- [4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. DOI: <https://doi.org/10.1145/2024716.2024718>
- [5] Xianmin Chen and Niraj K. Jha. 2016. Reducing wire and energy overheads of the SMART NoC using a setup request network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* PP, 99 (2016), 1–14. DOI: <https://doi.org/10.1109/TVLSI.2016.2538284>
- [6] William J. Dally and Brian Towles. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Annual Design Automation Conference (DAC'01)*. ACM, New York, 684–689. DOI: <https://doi.org/10.1145/378239.379048>
- [7] Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. 2009. Express cube topologies for on-chip interconnects. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 163–174.
- [8] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. 2009. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, 184–195. <http://doi.acm.org/10.1145/1555754.1555779>
- [9] John Kim, James Balfour, and William Dally. 2007. Flattened butterfly topology for on-chip networks. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'40)*. IEEE Computer Society, Washington, DC, 172–182. DOI: <https://doi.org/10.1109/MICRO.2007.15>
- [10] Tushar Krishna, Chia-Hsin Owen Chen, Woo-Cheol Kwon, and Li-Shiuan Peh. 2013. Breaking the on-chip latency barrier using SMART. In *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA 2013 (Shenzhen, China, Feb. 23-27, 2013))*. 378–389. DOI: <https://doi.org/10.1109/HPCA.2013.6522334>
- [11] Amit Kumar, Partha Kundu, Arvind P. Singh, Li-Shiuan Peh, and Niraj K. Jha. 2007. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *Proceedings of ICCD*. IEEE, 63–70. <http://dblp.uni-trier.de/db/conf/iccd/iccd2007.html#KumarKSPJ07>.
- [12] Amit Kumar, Li-Shiuan Peh, and Niraj K. Jha. 2008. Token flow control. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'41)*. IEEE Computer Society, Washington, DC, 342–353. DOI: <https://doi.org/10.1109/MICRO.2008.4771803>
- [13] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. 2007. Express virtual channels: Towards the ideal interconnection fabric. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, 150–161. DOI: <https://doi.org/10.1145/1250662.1250681>
- [14] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano, and Tsutomu Yoshinaga. 2009. Prediction router: Yet another low latency on-chip router architecture. In *Proceedings of HPCA*. IEEE Computer Society, 367–378. <http://dblp.uni-trier.de/db/conf/hpca/hpca2009.html#MatsutaniKAY09>.

- [15] Amirhossein Mirhosseini, Mohammad Sadrosadati, Behnaz Soltani, Hamid Sarbazi-Azad, and Thomas F. Wenisch. 2017. BiNoCHS: Bimodal network-on-chip for CPU-GPU heterogeneous systems. In *Proceedings of the 11th IEEE/ACM International Symposium on Networks-on-Chip*. ACM, 7.
- [16] Robert Mullins, Andrew West, and Simon Moore. 2004. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)*. IEEE Computer Society, Washington, DC, 188–197. <http://dl.acm.org/citation.cfm?id=998680.1006717>
- [17] Sunghyun Park, Tushar Krishna, Chia-Hsin Owen Chen, Bhavya K. Daya, Anantha Chandrakasan, and Li-Shiuan Peh. 2012. Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI. In *Proceedings of DAC*, Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun (Eds.). ACM, 398–405. <http://dblp.uni-trier.de/db/conf/dac/dac2012.html#ParkKCDP12>.
- [18] Li-Shiuan Peh and William J. Dally. 2001. A delay model and speculative architecture for pipelined routers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. IEEE Computer Society, Washington, DC, USA, 255–266. <http://dl.acm.org/citation.cfm?id=580550.876446>
- [19] Jacob Postman, Tushar Krishna, Christopher Edmonds, Li-Shiuan Peh, and Patrick Chiang. 2013. SWIFT: A low-power network-on-chip implementing the token flow control router architecture with swing-reduced interconnects. *IEEE Trans. Very Large Scale Integr. Syst.* 21, 8 (Aug. 2013), 1432–1446. <http://dx.doi.org/10.1109/TVLSI.2012.2211904>
- [20] Korey Sewell, Ronald G. Dreslinski, Thomas Manville, Sudhir Satpathy, Nathaniel Pinckney, Geoffrey Blake, Michael Cieslak, Reetuparna Das, Thomas F. Wenisch, Dennis Sylvester, et al. 2012. Swizzle-switch networks for many-core systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 2, 2 (2012), 278–294.
- [21] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the 2012 IEEE/ACM 6th International Symposium on Networks-on-Chip (NOCS'12)*. IEEE Computer Society, Washington, DC, 201–210. DOI : <https://doi.org/10.1109/NOCS.2012.31>
- [22] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpfen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. 2002. The RAW microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro* 22, 2 (2002), 25–35.
- [23] Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Priya Iyer, Arvind Singh, Tiju Jacob, and Shailendra Jain. 2007. An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS. In *Proceedings of the IEEE International Solid-State Circuits Conference*. 98–99.

Received December 2018; revised August 2019; accepted August 2019