Breaking the On-Chip Latency Barrier Using SMART

Tushar Krishna Chia-Hsin Owen Chen Woo Cheol Kwon Li-Shiuan Peh
Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute of Technology, Cambridge, MA 02139
*{tushar, owenhsin, wckwon, peh}@csail.mit.edu**

Abstract

As the number of on-chip cores increases, scalable on-chip topologies such as meshes inevitably add multiple hops in each network traversal. The best we can do right now is to design 1-cycle routers, such that the low-load network latency between a source and destination is equal to the number of routers + links (i.e. hops×2) between them. OS/compiler and cache coherence protocols designers often try to limit communication to within a few hops, since on-chip latency is critical for their scalability. In this work, we propose an on-chip network called SMART (Single-cycle Multi-hop Asynchronous Repeated Traversal) that aims to present a single-cycle data-path all the way from the source to the destination. We do not add any additional fast physical express links in the data-path; instead we drive the shared crossbars and links asynchronously up to multiple-hops within a single cycle. We design a router + link microarchitecture to achieve such a traversal, and a flow-control technique to arbitrate and setup multi-hop paths within a cycle. A place-and-routed design at 45nm achieves 11 hops within a 1GHz cycle for paths without turns (9 for paths with turns). We observe 5-8X reduction in low-load latencies across synthetic traffic patterns on an 8×8 CMP, compared to a baseline 1-cycle router. Full-system simulations with SPLASH-2 and PARSEC benchmarks demonstrate 27/52% and 20/59% reduction in runtime and EDP for Private/Shared L2 designs.

1. Introduction

Over the last decade, computer architects have been delivering higher FLOPS/cycle by increasing the number of on-chip cores, instead of increasing the clock frequency, because of the power wall. While increasing the number of cores is not very hard (due to Moore's Law), connecting these cores is. The reason is that more cores translates to more hops[1] to get from one core to another. Every hop adds an additional on-chip router (required at each hop to enable multiplexing of multiple flits over shared links) along the route, which increases the latency and energy overhead of every network traversal.

The equation for network latency (T) of a packet is [11]:

$$T = H \cdot t_r + H \cdot t_w + T_c + L/b \quad (1)$$

H is the number of hops, t_r is the router pipeline delay, t_w is the wire (between two routers) delay, T_c is the contention delay at routers, and L/b is the serialization delay for the body and tail flits, i.e. time for a packet of length L to cross a channel with bandwidth b.

One proposed approach to reduce this latency is topology, by using high-radix routers [13, 19, 8, 22, 34]. The idea is to reduce H, by adding explicit links between physically distant routers, thus reducing the number of routers on the route. However, this is done at the cost of thinner channels (i.e. smaller b) which increases serialization delay. Moreover, higher number of input/output ports at routers leads to increased complexity of the routing, allocation and crossbar blocks, increasing router delay t_r and router power. Instead, most commercial and research multicore prototypes [15, 16, 36, 1] have opted for simpler topologies like rings and meshes to ease design (layout and verification) and reduce router delay and energy. Network latency in such systems has been mitigated by shrinking t_r to 1, using router microarchitectural and flow-control techniques [24, 27, 32, 23, 25, 31, 30]. As core count increases though, H inevitably increases. Average hop counts in a $k \times k$ mesh increase linearly with k. As we design 1024 core chips [2, 28, 12, 18] for the exascale era, high hop counts will lead to horrendous on-chip network traversal latency and energy creating a stumbling block to core count scaling.

How critical is on-chip latency for overall system performance? We compare three networks: (1) 3-cycle router, i.e. $t_r = 3$ (modeled similar to Intel's recent 48-core SCC [16]), (2) 1-cycle router, i.e. $t_r = 1$ (the state-of-the-art in academic literature to date, described in Section 2), and (3) an ideal 1-cycle network, i.e. $T = 1 + L/b$ (every flit is magically sent from the source NIC to its destination NIC after 1-cycle with zero contention, which essentially implies that every core is 1-hop away). t_w (i.e. wire delay per hop) is assumed to be 1

*The authers acknowledge the support of DARPA UHPC, SMART LEES, and MARCO C-FAR. A special thanks to Sunghyun Park from MIT and Michael Pellauer from Intel for highly useful discussions on the SMART interconnect and pipeline respectively.

[1] We define **hop** to be the physical distance between neighboring tiles, which is typically 1-2mm [14, 15]. In this paper, 1-hop = 1mm based on place-and-route of a Freescale PowerPC e200z7 core in 45nm.

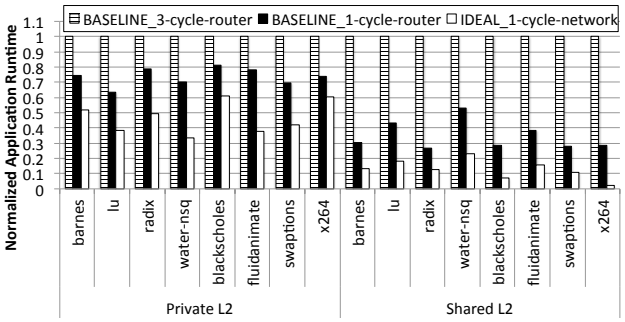


Figure 1: Impact of on-chip latency on full-system runtime

for (1) and (2). We perform full-system simulations on a 64-core system[2] laid out as a mesh, and look at runtime across a suite of SPLASH-2 [3] and PARSEC [9] benchmarks for both a Private and a (distributed) Shared L2 configuration. Figure 1 shows 26% and 52% reduction in runtime on average for (2) and (3) compared to (1) for a Private L2 design, where only L2 misses traverse the network. For a Shared L2 design, both L1 and L2 misses traverse the network, making network latency more critical, which is reflected by a 63% and 85% runtime reduction for (2) and (3) compared to (1).

A high on-chip latency not just delays requests and responses, but also slows down the injection of other requests and responses (due to dependencies), leading to poorer throughput and overall system slowdown. This is the reason why coherence protocol designers prefer Private L2 designs, while programmers and compiler/OS designers try to map data close to sharers to minimize the average network hops. However, there is only so much that the protocol or software can do since a core has limited 1-hop neighbors.

In this work, we present a solution to approach the ideal 1-cycle network for any source-destination pair in a mesh. Our proposed NoC is named SMART, for Single-cycle Multi-hop Asynchronous Repeated Traversal. As the name suggests, we embed *asynchronous repeaters* within each router's crossbar, and size them to drive signals up to multiple hops within a single-cycle. We optimize network latency as follows:

$$T = (H/HPC) \cdot t_r + (H/HPC) \cdot t_w + T_c + L/b \qquad (2)$$

where *HPC* stands for number of Hops Per Cycle. We reduce the effective number of hops to $\lceil (H/HPC) \rceil$, without adding any additional physical wires in the data-path or reducing b like the high-radix router solutions do.

This paper makes the following contributions:

- We advocate for a single-cycle traversal across multiple routers in a network.
- A single-cycle multi-mm interconnect circuit is presented, integrated into a regular mesh topology.
- A network flow-control mechanism is presented that enables flits to setup arbitrary multi-hop paths (with turns) within a cycle, and then traverse them within a cycle.

On a 64-core mesh, synthetic traffic shows 5-8X reduction in average network latency, while full-system SPLASH-2 and PARSEC traffic shows 27/52% reduction in average runtime for Private/Shared, compared to a 1-cycle router.

The paper is organized as follows. Section 2 describes the baseline 1-cycle router. Section 3 introduces the SMART link, and how it is embedded into a router. Section 4 demonstrates the design for a k-ary 1-Mesh, and Section 5 extends it to a k-ary 2-Mesh. Section 6 describes implementation details, up to layout. Section 7 presents our evaluations. Section 8 contrasts against prior art and Section 9 concludes.

2. Background

Networks-on-Chip (NoCs) consist of shared links, with routers at crosspoints. Routers perform multiplexing of flits on the links, and buffer flits in case of contention. Each hop consists of a router + link traversal. A router performs the following actions [11]:

Buffer Write (BW): The incoming flit is buffered.

Route Compute (RC): The incoming head flit chooses an output port to depart from.

Switch Allocation (SA): Buffered flits arbitrate among themselves for the crossbar switch. At the end of this stage, there is at most one winner for every input and output port of the crossbar.

VC Selection (VS): Head flits that win SA reserve a VC for the next router, from a pool of free VCs [26].

The winners of SA proceed to **Switch Traversal (ST)** and **Link Traversal (LT)** to the next router.

Plethora of research in NoCs over the past decade coupled with technology scaling has allowed the actions within a router to move from serial execution to parallel execution, via lookahead routing [11], simplified VC selection [26], speculative switch arbitration [31, 30], non-speculative switch arbitration via lookaheads [24, 27, 32, 23, 25] to bypass buffering and so on. This has allowed the router delay t_r (Equation 1) to drop from 3-5 cycles in industry prototypes [15, 16] to 1-cycle in academic NoC-only prototypes [25, 32]. We use this state-of-the-art 1-cycle router as our baseline. ST and LT can be done together within a cycle [16, 32] giving us t_w = 1. Thus our baseline incurs 2-cycles-per-hop, and is shown in Figure 2. In case of contention, flits have to get buffered and could wait multiple cycles before they win SA and VS, increasing T_c, as shown at Router$_{n+i}$.

3. The SMART Interconnect

Adding asynchronous repeaters[3] is a standard way of reducing wire delay [20, 33]. We perform place-and-route for a 128-bit repeated wire in a commercial 45nm SOI technology using Cadence Encounter. We keep increasing the length of the wire, letting the tool size the repeaters appropriately, till it fails timing closure at 1ns (i.e. 1GHz). Figure 3 shows that a place-and-routed repeated wire in 45nm can

[2] Refer to Section 7 for methodology and configurations.

[3] A pair of inverters.

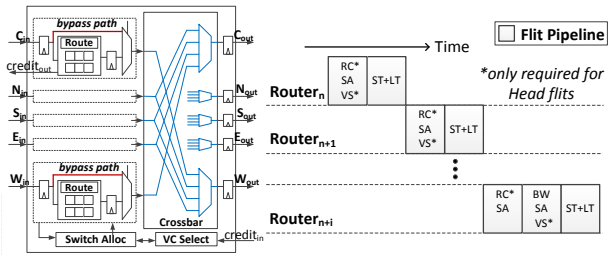


Figure 2: Baseline 1-cycle Router (2-cycles-per-hop)

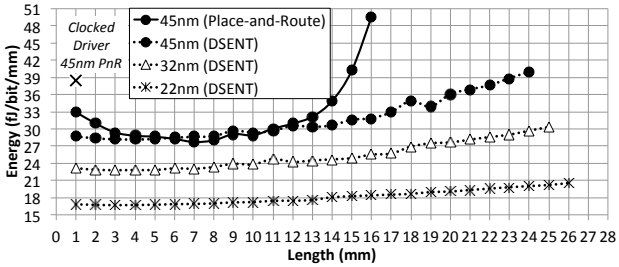


Figure 3: Achievable HPC_{max} for Repeated Links at 1GHz.
Wire Width: DRC_{min}, Wire Spacing: $3 \cdot DRC_{min}$, Metal Layer: M6. Repeater Spacing: 1mm

go up to 16mm in a ns[4]. We define the maximum length in *mm* that can be traversed in a cycle as HPC_{max}. Figure 3 shows a similar trend for HPC_{max} at 32nm and 22nm, with energy going down by 19% and 42% respectively, using the timing-driven NoC power modeling tool DSENT [35][5].

Router logic delay limits the network frequency to 1-2GHz at 45nm [16, 32]. Link drivers are accordingly sized to drive only 1mm (1-hop) in 0.5-1ns, before the signal is latched at the next router. SMART removes this constraint of latching signals at every hop. We exploit the positive slack in the link traversal stage by replacing clocked link drivers by asynchronous repeaters at every hop, thus driving signals HPC_{max}-hops within a cycle. HPC_{max} is a design-time parameter, which can be inferred from Figure 3. If we choose a 2mm tile size, or a 2GHz frequency, HPC_{max} will go down by half. Asynchronous repeaters also consume 14.3% lower energy/bit/mm than conventional clocked drivers, as shown in Figure 3, giving us a win-win. SMART is a better solution for exploiting the slack than deeper pipelining of the router with a higher clock frequency (e.g. Intel's 80-core 5GHz 5-stage router [15]) which, even if it were possible to do, does not reduce traversal latency (only improves throughput), and adds huge power overheads due to pipeline registers.

Figure 4a shows a SMART router. For simplicity, we only show $Core_{in}(C_{in})$[6], $West_{in}(W_{in})$ and $East_{out}(E_{out})$ ports. All other input ports are identical to W_{in}, and all other output ports are identical to E_{out}. Each repeater has to be sized to drive not just the link, but also the muxes (2:1 bypass and 4:1 Xbar) at the next router, before a new repeater is encountered. Using the same methodology with Cadence Encounter, this reduces HPC_{max} to 11 at 1GHz (Section 6).

Figure 4a shows the three primary components of the design: (a) Buffer Write enable (BW_{ena}) at the input flip flop which determines if the input signal is latched or not, (2) Bypass Mux select (BM_{sel}) at the input of the crossbar to choose between the local buffered flit, and the bypassing flit on the link, and (3) Crossbar select (XB_{sel}). Figure 4b shows an example of a multi-hop traversal: a flit from Router R0 traverses 3-hops within a cycle, till it is latched at R3. The crossbars at R1 and R2 are preset to connect the W_{in} to E_{out}, with their BM_{sel} preset to choose bypass over local. A SMART path can thus be created by appropriately setting BW_{ena}, BM_{sel}, and XB_{sel} at intermediate routers. In the next section, we describe the flow control to preset these signals.

4. SMART in a *k*-ary 1-Mesh

Table 1 defines terminology that will be used throughout the paper. We start by demonstrating how SMART works in a *k*-ary 1-Mesh, shown in Figure 5. Each router has 3 ports: West, East and Core[7]. As shown earlier in Figure 4a, E_{out}_xb can be connected either to C_{in}_xb or W_{in}_xb. W_{in}_xb can be driven either by *bypass*, *local* or 0, depending on BM_{sel}.

The design is called **SMART_1D** (since routers can be bypassed only along one dimension). The design will be extended to a *k*-ary 2-Mesh to incorporate turns, in Section 5. For purposes of illustration, we will assume HPC_{max} to be 3.

4.1. SMART-hop Setup Request (SSR)

The SMART router pipeline is shown in Figure 6. A SMART-hop starts from a start router, where flits are buffered. Unlike the baseline router, Switch Allocation in SMART occurs over two stages: Switch Allocation Local (SA-L) and Switch Allocation Global (SA-G). SA-L is identical to the SA stage in the conventional pipeline (described earlier in Section 2): every start router chooses a winner for each output port from among its buffered (local) flits. In the next cycle, instead of the winners directly traversing the crossbar (ST), they broadcast a SMART-hop setup request (*SSR*) via *dedicated* repeated wires (which are inherently multi-drop[8]) up to HPC_{max}. These dedicated *SSR* wires are shown in Figure 5. These are $log_2(1+HPC_{max})$-bits wide, and are part of the control-path. The *SSR* carries the length (in hops) up to which the flit winner wishes to go. For instance, *SSR* = 2 indicates a 2-hop path request. Each flit tries

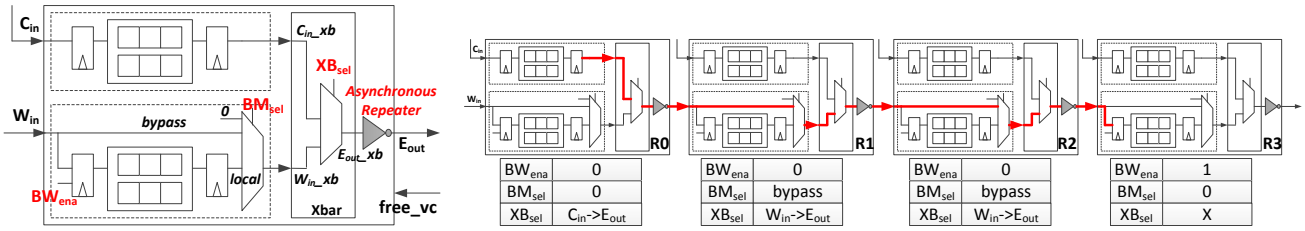
[4]The sharp rise in energy past 13mm can be attributed to a limitation of the place-and-route tool, which zig-zags wires to fit to a fixed global grid, that is unfortunately not a multiple of M6 width, adding unnecessary wire length. A custom design can potentially go farther and with flatter energy profile.

[5]DSENT's HPC_{max} projections are slightly overestimated because it does not model inter-layer via parasitics (needed to access the repeater transistors on M1 from the link on M6), which become significant when there are many repeaters.

[6]C_{in} does not have a bypass path like the other ports because all flits from the NIC have to get buffered at the first router, before they can create SMART paths, which will be explained later in Section 4.

[7]For illustration purposes, we only show C_{in}, W_{in} and E_{out} in the figures.

[8]Wire cap is an order of magnitude higher than gate cap, adding no overhead if all nodes connected to the wire receive.



(a) SMART Router Microarchitecture (b) Single-cycle Multi-hop Asynchronous Repeated Traversal Example

Figure 4: SMART: Single-cycle Multi-hop Asynchronous Repeated Traversal

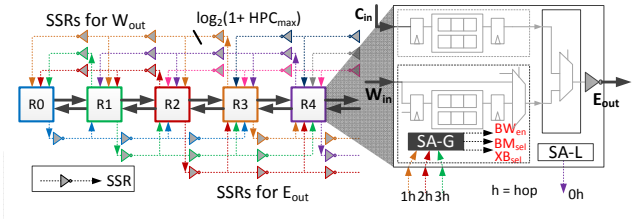


Figure 5: k-ary 1-Mesh with dedicated SSR links.

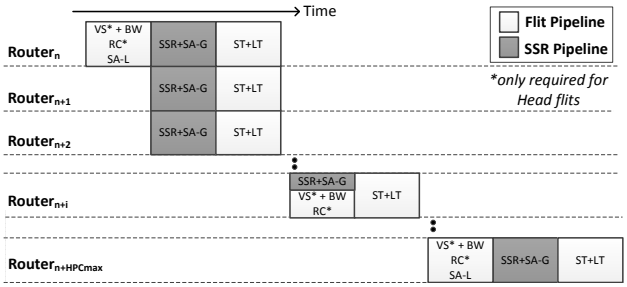


Figure 6: SMART Pipeline

to go as close as possible to its ejection router, hence $SSR = \min(HPC_{max}, H_{remaining})$.

During SA-G, all inter routers arbitrate among the *SSRs* they receive, to set the BW_{ena}, BM_{sel} and XB_{sel} signals. The arbiters guarantee that only *one* flit will be allowed access to any particular input/output port of the crossbar. In the next cycle (ST+LT), SA-L winners that also won SA-G at their start routers traverse the crossbar and links upto multiple hops till they are stopped by BW_{ena} at some router. Thus flits spend at least 2 cycles (SA-L and SA-G) at a start router before they can use the switch. Flits can end up getting prematurely stopped (i.e before their *SSR* length) depending on the SA-G results at different routers. *SSR* traversal and SA-G occur serially (see Section 6 for timing implications).

We illustrate all these with examples. In Figure 7, Router R2 has $Flit_A$ and $Flit_B$ buffered at C_{in}, and $Flit_C$ and $Flit_D$ buffered at W_{in}, all requesting E_{out}. Suppose $Flit_D$ wins SA-L during Cycle-0. In Cycle-1, it sends out $SSR_D = 2$ (i.e. request to stop at R4) out of E_{out} to Routers R3, R4 and R5. SA-G is performed at each router. At R2, which is 0-hops away ($< SSR_D$), BM_{sel} = local, $XB_{sel} = W_{in}\_xb \rightarrow E_{out}\_xb$. At R3, which is 1-hop away ($< SSR_D$), BM_{sel} = bypass, $XB_{sel} = W_{in}\_xb \rightarrow E_{out}\_xb$. At R4, which is 2-hops away ($= SSR_D$), BW_{ena} = high. At R5, which is 3-hops away ($> SSR_D$), SSR_D is ignored. In Cycle-2, $Flit_D$ traverses the crossbars and links at R2 and R3, and is stopped and buffered at R4.

What happens if there are competing *SSRs*? In the same example, suppose R0 also wants to send $Flit_E$ 3-hops away to R3, as shown in Figure 8. In Cycle-1, R2 sends out SSR_D as before, and in addition R0 sends $SSR_E = 3$ out of E_{out} to R1, R2 and R3. Now at R2 there is a conflict between SSR_D and SSR_E for the W_{in}_xb and E_{out}_xb ports of the crossbar. *SA-G priority* decides which *SSR* wins the crossbar. More details about priority will be discussed later

Table 1: Terminology

Term	Meaning
HPC	Hops Per Cycle. The number of hops traversed in a cycle by any flit.
HPC$_{max}$	Maximum number of hops that can be traversed in a cycle by a flit. This is fixed at design time.
SMART-hop	*Multi-hop* path traversed in a *Single-cycle* via a SMART link. It could be straight, or have turns. The length of a SMART-hop can vary anywhere from 1-hop to HPC_{max}.
injection router	First router on the route. The source NIC injects a flit into the C_{in} port of this router.
ejection router	Last router on the route. This router ejects a flit out of the C_{out} port to the destination NIC.
start router	Router from which any SMART-hop starts. This could be the injection router, or any router along the route.
inter router	Any intermediate router on a SMART-hop.
stop router	Router at which any SMART-hop ends. This could be the ejection router or any router along the route.
turn router	Router at a turn (W_{in}/E_{in} to N_{out}/S_{out}, or N_{in}/S_{in} to W_{out}/E_{out}) along the route.
local flits	Flits buffered at any start router.
bypass flits	Flits which are bypassing inter routers.
SMART-hop Setup Request (SSR)	Length (in hops) for a requested SMART-hop. For example, *SSR=H* indicates a request to stop H-hops away. Optimization: Additional *ejection*-bit if requested stop router is ejection router.
premature stop	A flit is forced to stop before its requested *SSR* length.
Prio=Local	Local flits have higher priority over bypass flits, i.e. Priority α 1/(hops_from_start_router).
Prio=Bypass	Bypass flits have higher priority over local flits, i.e. Priority α (hops_from_start_router).
SMART_1D	Design where routers along the dimension (both X and Y) can be bypassed. Flits need to stop at the turn router.
SMART_2D	Design where routers along the dimension and one turn can be bypassed.

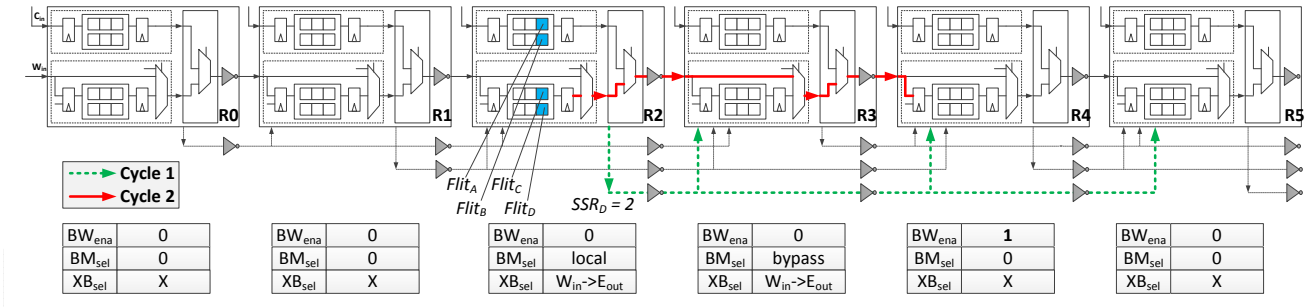


Figure 7: SMART Example: No *SSR* Conflict

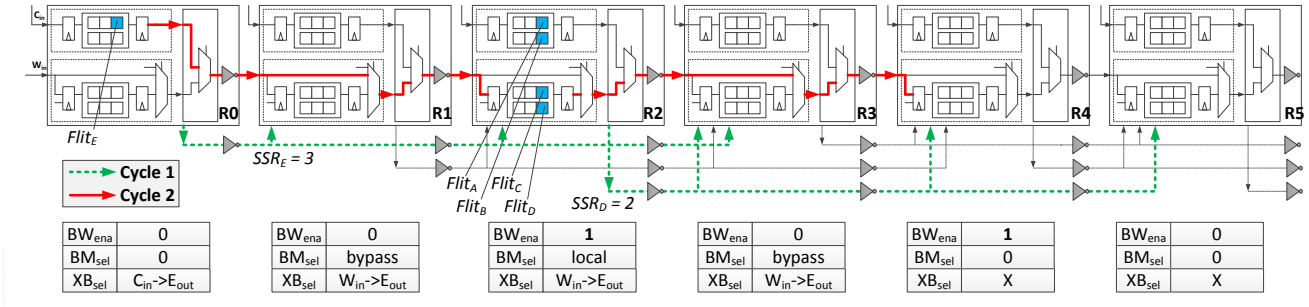


Figure 8: SMART Example: *SSR* Conflict with Prio=Local

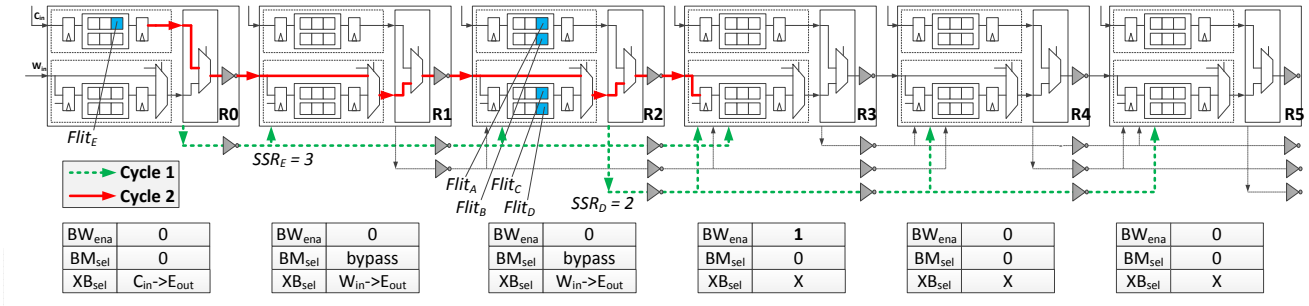


Figure 9: SMART Example: *SSR* Conflict with Prio=Bypass

in Section 4.2. For now, let us assume Prio=Local (which is defined in Table 1) so $Flit_E$ loses to $Flit_D$. The values of BW_{ena}, BM_{sel} and XB_{sel} at each router for this priority are shown in Figure 8. In Cycle-2, $Flit_E$ traverses the crossbar and link at R0 and R1, but is stopped and buffered at R2. $Flit_D$ traverses the crossbars and links at R2 and R3 and is stopped and buffered at R4. $Flit_E$ now goes through BW and SA-L at R2 before it can send a new *SSR* and continue its network traversal. A free VC/buffer is guaranteed to exist whenever a flit is made to stop (see Section 4.4).

4.2. Switch Allocation Global: Priority

Figure 9 shows the previous example with Prio=Bypass instead of Prio=Local. This time, in Cycle-2, $Flit_E$ traverses all the way from R0 to R3, while $Flit_D$ is stalled.

Do all routers need to enforce the same priority? Yes. This guarantees that all routers will arrive at the same consensus about which *SSRs* win and lose. This is required for correctness. In the example discussed earlier in Figures 8 and 9, BW_{ena} at R3 was low with Prio=Local, and high with Prio=Bypass. Suppose R2 performs Prio=Bypass, but R3 performs Prio=Local, $Flit_E$ will end up going from R0 to R4, instead of stopping at R3. This is not just a misrouting issue, but also a signal integrity issue because HPC_{max} is 3, but the flit was forced to go up to 4 hops in a cycle, and will not be able to reach the clock edge in time. Note that enforcing the same priority is only necessary for SA-G, which corresponds to the global arbitration among SA-L winners at every router. During SA-L, however, different routers/ports can still choose to use different arbiters (round robin, queueing, priority) depending on the desired QoS/ordering mechanism.

Can a flit arrive at a router, even though the router is not expecting it (i.e. false positive[9])? No. All flits that arrive at a router are expected, and will stop/bypass based on the success of their *SSR* in the previous cycle. This is guaranteed since all routers enforce the same SA-G priority.

Can a flit not arrive at a router, even though the router is expecting it (i.e. false negative)? Yes. It is possible for the router to be setup for stop/bypass for some flit, but no flit

[9]The result of SA-G (BW_{ena}, BM_{sel} and XB_{sel}) at a router is a prediction for the null hypothesis: a flit will arrive the next cycle, and stop/bypass.

arrives. This can happen if that flit is forced to prematurely stop earlier due to some *SSR* interaction at prior inter routers that the current router is not aware of. For example, suppose a local flit at W_{in} at R1 wants to eject out of C_{out}. A flit from R0 will prematurely stop at R1's W_{in} port if Prio=Local is implemented. However, R2 will still be expecting the flit from R0 to arrive[10]. Unlike false positives, this is not a correctness issue but just a performance (throughput) issue, since some links go idle which could have potentially been used by other flits if more global information were available.

4.3. Ordering

In SMART, any flit can be prematurely stopped based on the interaction of *SSRs* that cycle. We need to ensure that this does not result in re-ordering between (a) flits of the same packet, or (b) flits from the same source (if Pt-to-Pt ordering is required in the coherence protocol).

The first constraint is in routing (relevant to 2D topologies). Multi-flit packets, and Pt-to-Pt ordered virtual networks should only use deterministic routes, to ensure that prematurely buffered flits do not end up choosing alternate routes, while bypassing flits continue on the old route.

The second constraint is in SA-G priority. Every input port has a bit to track if there is a prematurely stopped flit among its buffered flits. When a *SSR* is received at an input port, and there is either (a) a prematurely buffered Head/Body flit, or (b) a prematurely buffered flit within a Pt-to-Pt ordered virtual network, the incoming flit is stopped.

4.4. Guaranteeing free VC/buffers at stop routers

In a conventional network, a router's output port tracks the IDs of all free VCs at the neighbor's input port. A buffered Head flit chooses a free VCid for its *next* router (neighbor), before it leaves the router. The neighbor signals back when that VCid becomes free. In a SMART network, the challenge is that the *next* router could be any router that can be reached within a cycle. A flit at a start router choosing the VCid before it leaves will not work because (a) it is not guaranteed to reach its presumed next router, and (b) multiple flits at different start routers might end up choosing the same VCid. Instead, we let the VC selection occur at the stop router. Every SMART router receives 1-bit from each neighbor to signal if *at least one* VC is free[11]. During SA-G, if an *SSR* requests an output port where there is no free VC, BW_{ena} is made high and the corresponding flit is buffered. This solution does not add any extra multi-hop wires for VC signaling. The signaling is still between neighbors. Moreover, it ensures that a Head flit comes into a router's input port only if that input port has free VCs, else the flit is stopped at the previous router.

However, this solution is conservative because a flit will be stopped prematurely if the neighbor's input port does not have free VCs, even if there was no competing *SSR* at the neighbor and the flit would have bypassed it without having to stop.

How do Body/Tail flits identify which VC to go to at the stop router? Using their *injection_router* id. Every input port maintains a table to map a VCid to an injection router id[12]. Whenever the Head flit is allocated a VC, this table is updated. The injection router id entry is cleared when the Tail arrives. The VC is freed when the Tail leaves. We implement private buffers per VC, with depth equal to the maximum number of flits in the packet (i.e. virtual cut-through), to ensure that the Body/Tail will always have a free buffer in its VC[13].

What if two Body/Tail flits with same injection_router id arrive at a router? We guarantee that this will never occur by forcing *all* flits of a packet to leave from an output port of a router, before flits from another packet can leave from that output port (i.e virtual cut-through). This guarantees a unique mapping from injection router id to VCid in the table at every router's input port.

What if a Head bypasses, but Body/Tail is prematurely stopped? The Body/Tail still needs to identify a VCid to get buffered in. To ensure that it does have a VC, we make the Head flit reserve a VC not just at its stop router, but also at all its inter routers, even though it does not stop there. This is done from the *valid*, *type* and *injection_router* fields of the bypassing flit. The Tail flit frees the VCs at all the inter routers. Thus, for multi-flit packets, VCs are reserved at all routers, just like the baseline. But the advantage of SMART is that VCs are reserved and freed at *multiple routers* within the *same cycle*, thus reducing the buffer turnaround time.

4.5. Additional Optimizations

We add additional optimizations to SMART to push it towards an ideal 1-cycle network described in Section 1.

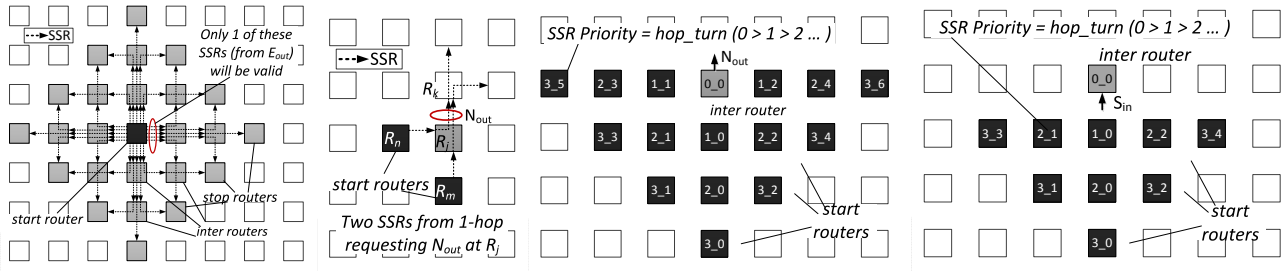
Bypassing the ejection router. So far we have assumed that a flit starting at an injection router traverses one (or more) SMART-hops till the ejection router, where it gets buffered and requests for the C_{out} port. We add an extra $ejection$-bit in the *SSR* to indicate if the requested stop router corresponds to the ejection router for the packet, and not any intermediate router on the route. If a router receives a *SSR* from H-hops away with value H (i.e. request to stop there), $H < HPC_{max}$, and the *ejection*-bit is high, it arbitrates for C_{out} port during SA-G. If it loses, BW_{ena} is made high.

[10] The *valid*-bit from the flit is thus used in addition to BW_{ena} when deciding whether to buffer.

[11] If the router has multiple virtual networks (vnets) for the coherence protocol, we need a 1-bit free VC signal from the neighbors for each vnet. The *SSR* also needs to carry the vnet number, so that the inter routers can know which vnet's free VC signal to look at.

[12] The table size equals the number of multi-flit VCs at that input port.

[13] Extending this design to fewer buffers than the number of flits in a packet would involve more signaling, and is left for future work.



(a) k-ary 2-Mesh with SSR wires from shaded start router. (b) Conflict between two SSRs for N_{out} port. (c) Fixed Priority at N_{out} port of inter router. (d) Fixed Priority at S_{in} port of inter router.

Figure 10: SMART_2D: $SSRs$ and their SA-G priorities.

Bypassing SA-L at low load. We add no-load bypassing [11] to the SMART router. If a flit comes into a router with an empty input port and no SA-L winner for its output port for that cycle, it sends *SSRs* directly, in parallel to getting buffered, without having to go through SA-L. This reduces t_r at lightly-loaded start routers to 2, instead of 3, as shown in Figure 6 for Router$_{n+i}$. Multi-hop traversals within a single-cycle meanwhile happen at all loads.

4.6. Summary

In summary, a SMART NoC works as follows:

- Buffered flits at injection/start routers arbitrate locally to choose input/output port winners during SA-L.
- SA-L winners broadcast *SSRs* along their chosen routes, and each router arbitrates among these *SSRs* during SA-G.
- SA-G winners traverse multiple crossbars and links asynchronously within a cycle, till they are explicitly stopped and buffered at some router along their route.

In a **SMART_1D** design with both ejection and no-load bypass enabled, if HPC_{max} is larger than the maximum hops in any route, a flit will only spend 2 cycles in the entire network in the best case (1-cycle for *SSR* and 1-cycle for ST+LT all the way to the destination NIC).

5. SMART in a k-ary 2-Mesh

We demonstrate how SMART works in a k-ary 2-Mesh. Each router has 5 ports: West, East, North, South and Core.

5.1. Bypassing routers along dimension

We start with a design where we do not allow bypass at turns, i.e. all flits have to stop at their turn routers. We re-use **SMART_1D** described for a k-ary 1-Mesh in a k-ary 2-Mesh. The extra router ports only increase the complexity of the SA-L stage, since there are multiple local contenders for each output port. Once each router chooses SA-L winners, SA-G remains identical to the description in Section 4.1. The E_{out}, W_{out}, N_{out} and S_{out} ports have dedicated *SSR* wires going out till HPC_{max} along that dimension. Each input port of the router can receive only one *SSR* from a router that is H-hops away. The *SSR* requests a stop, or a bypass along that dimension. Flits with turning routes perform their traversal one-dimension at a time, trying to bypass as many routers as possible, and stopping at the turn routers.

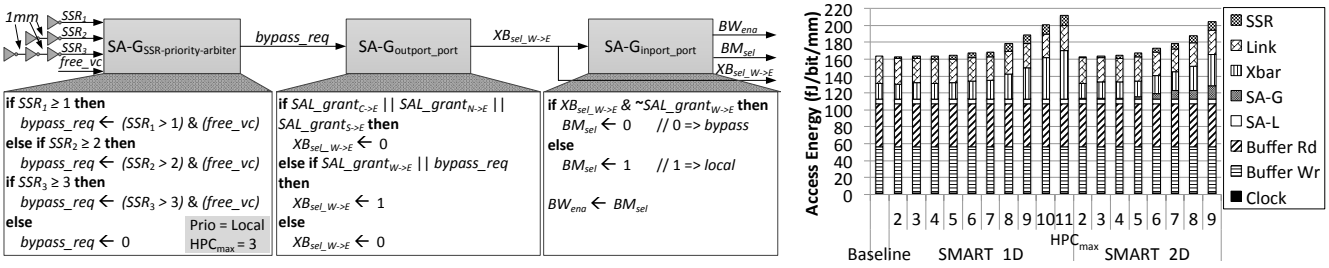
5.2. Bypassing routers at turns

In a k-ary 2-Mesh topology, all routers within a HPC_{max} neighborhood can be reached within a cycle, as shown in Figure 10a by the shaded diamond. We now describe **SMART_2D** which allows flits to bypass both the routers along a dimension and the turn router(s). We add dedicated *SSR* links for each possible XY/YX path from every router to its HPC_{max} neighbors. Figure 10a shows that the E_{out} port has 5 *SSR* links, in comparison to only one in the SMART_1D design. During the routing stage, the flit chooses one of these possible paths. During the SA-G stage, the router broadcasts *one SSR* out of each output port, on one of these possible paths. We allow only one turn within each HPC_{max} quadrant to simplify the *SSR* signaling.

SA-G Priority. In the SMART_2D design, there can be more than one *SSR* from H-hops away, as shown in the example in Figure 10b for router R_j. R_j needs a specific policy to choose between these requests, to avoid sending false positives on the way forward to R_k. Section 4.2 discussed that false positives can result in misrouted flits or flits trying to bypass beyond HPC_{max}, thus breaking the system. To arbitrate between *SSR*s from routers that are the same number of hops away, we choose Straight > Left Turn > Right Turn. For the inter router R_j in Figure 10b, the *SSR* from R_m will have higher priority (1_0) over the one from R_n (1_1) for the N_{out} port, as it is going straight, based on Figure 10c. Similarly at R_k, the *SSR* from R_m will have higher priority (2_0) over the one from R_n (2_1) for the S_{in} port, based on Figure 10d. Thus both routers R_j and R_k will unambiguously prioritize the flit from R_m to use the links, while the flit from R_n will stop at Router R_j. Any priority scheme will work as long as every router enforces the same priority.

6. SMART Implementation

In this section, we describe the implementation details of SMART_1D and SMART_2D designs, and discuss overheads. All numbers are for a k-ary 2-Mesh, i.e. the crossbar has 5-ports (with u-turns disallowed).



(a) Implementation of SA-G at W_{in} and E_{out} (Figure 4a) for SMART_1D

(b) Energy/Access (i.e. Activity = 1) for each bit sent

Figure 11: SMART Implementation

The SMART data-path, shown earlier in Figure 4, is modeled as a series of 128-bit 2:1 mux (for bypass) followed by a 4:1 mux (crossbar), followed by a 128-bit 1mm link.

The SMART control-path consists of HPC_{max}-hops repeated wire delay (*SSR* traversal), followed by logic gate delay (SA-G). In SMART_1D, each input port receives one *SSR* from every router up to HPC_{max}-hops away in that dimension. The logic for SA-G for Prio=Local in a SMART_1D design at the W_{in} and E_{out} ports of the router is shown in Figure 11a[14]. The input and output signals correspond to the ones shown in the router in Figure 4a[15].

In SMART_2D, all routers that are H-hops away, $H \in [1, HPC_{max}]$, together send a total of $(2 \times HPC_{max} - 1)$ *SSRs* to every input port. SA-$G_{SSR\_priority\_arbiter}$ is similar to Figure 11a in this case and chooses a set of winners based on hops, while SA-G_{output_port} disambiguates between them based on turns, as discussed earlier in Section 5.2.

We choose a clock frequency of 1GHz based on SA-L critical path in the baseline 1-cycle router at 45nm [32]. We design each of the SMART components in RTL, run it through synthesis and layout for increasing HPC_{max} values, till timing fails at 1GHz. This gives us energy and area numbers for every HPC_{max} design point. We incorporate these into energy and area numbers for the rest of the router components from DSENT [35]. Figure 11b plots the energy/bit/hop for accessing each component. For instance, if a flit wins SA-L and SA-G and traverses a SMART-hop of length 4 in an HPC_{max}=8 design, the energy consumed will be $E_{SA-L} + 8 \cdot E_{SSR} + 4 \cdot E_{SA-G} + E_{buf\_rd} + 4 \cdot E_{Xbar} + 4 \cdot E_{Link} + E_{buf\_wr}$.

The SMART data-path is able to achieve a HPC_{max} of 11. The extra energy consumed by the repeaters for driving the bypass and crossbar muxes is part of the Xbar component in Figure 11b, and increases comparatively insignificantly till about HPC_{max}=8, beyond which it shows a steep rise, consuming 3X of the baseline Xbar energy at HPC_{max}=11. The total data-path (Xbar+Link) energy for HPC_{max}=11 goes up by 35fJ/bit/hop, compared to the baseline. However, compared to the buffer energy (110fJ/bit/hop) that will be saved with the additional bypassing brought about by longer HPC_{max}, and coupled with additional network latency savings along with further reduction of data-path energy per bit as technology scales, we believe it will be worthwhile to go with higher HPC_{max} as we scale to hundreds or a thousand cores. The repeaters do not add any area overhead since the crossbar area is wire dominated.

SMART_1D's control-path is able to achieve a HPC_{max} of 13 (890ps *SSR*, 90ps SA-G). But the overall HPC_{max} gets limited to 11 by the data-path. SA-G adds less than 1% of energy or area overhead.

SMART_2D's control-path is able to achieve a HPC_{max} of 9 (620ps *SSR*, 360ps SA-G), at which point the energy and area overheads go up to 8% and 5% respectively, due to the quadratic scaling of input *SSRs* with HPC_{max}. However, not all the input *SSR*s are likely to be active every cycle.

The total number of *SSR*-bits entering an input port are of $O(HPC_{max} \cdot log_2(HPC_{max}))$ and $O(HPC_{max}{}^2 \cdot log_2(HPC_{max}))$ in SMART_1D and SMART_2D respectively. But these do not affect tile area. However, the *SSRs* add energy overheads due to HPC_{max}-mm signaling whenever a *SSR* is sent.

Based on the energy results, we choose HPC_{max}=8 for both SMART_1D and SMART_2D for our evaluations. For SMART_1D, HPC_{max}=8 allows bypass of all routers along the dimension and the ejection, in our target 8-ary 2-Mesh.

7. Evaluation

We use the GEMS [29] + Garnet [5] infrastructure for all our evaluations, which provides a cycle-accurate timing model. Full-system simulations use Wind River Simics [4]. Network energy is calculated using DSENT [35] and our place-and-route results from Section 6. Our target system is shown in Table 2. The baseline design in all our runs is a state-of-the-art 1-cycle router described earlier in Section 2. All SMART designs are named as **SMART-HPC_{max}_1D/2D**. Prio=Local is assumed unless explicitly mentioned.

[14]The implementation of Prio=Bypass is not discussed but is similar.

[15]To reduce the critical path, BW_{ena} is relaxed such that it is 0 only when there are bypassing flits (since the flit's valid-bit is also used to decide when to buffer), and BM_{sel} is relaxed to always pick local if there is no bypass. XB_{sel} is strict and does not connect an input to an output port unless there is a local or *SSR* request for it.

Table 2: Target System and Configuration

Process		**On-chip Network**	
Technology	45nm	Topology	8-ary 2-Mesh
V_{dd}	1.0 V	Router Ports	5
Frequency	1.0 GHz	Routing	XY
Link Length	1mm	Flit Width	128-bit

Synthetic Traffic	
Virtual Channels	12 [1-flit/VC]

Full-system Traffic	
Processors	64 in-order SPARC
L1 Caches	Private 32kB I&D
L2 Caches	Private/Shared 1MB per core
Cache Coherence	MOESI distributed directory
Virtual Networks	3 (req, fwd, resp)
Virtual Channels	4 (req), 4 (fwd) [1-flit/VC], 4 (resp) [5-flit/VC]

7.1. Synthetic Traffic

7.1.1. SMART across different traffic patterns. We start by running SMART with synthetic traffic patterns. In the interest of space, we only show three of these in Figure 12. We compare 3 SMART designs: SMART-8_1D and SMART-8_2D (which are both achievable designs as discussed in Section 6), and SMART-15_2D which reflects the best that SMART can do in an 8×8 Mesh (with maximum possible hops = 15). We inject 1-flit packets to first understand the benefits of SMART without secondary effects due to flit serialization, and VC allocation across multiple routers etc. For the same reason, we also give enough VCs (12, derived empirically) to allow both the baseline and SMART to be limited by links, rather than VCs for throughput.

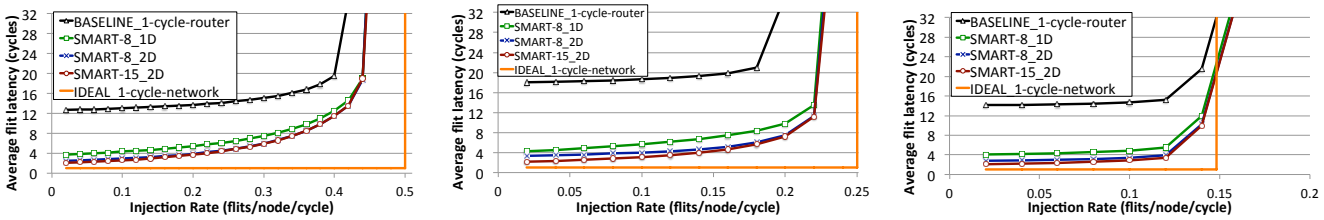
The striking feature about SMART from Figure 12 is that it pushes low-load latency to 4 and 2 cycles, for SMART_1D and SMART_2D respectively, across all traffic patterns, unlike the baseline where low-load latency is a function of the average hops, thus truly breaking the locality barrier. SMART-8_2D achieves most of the benefit of SMART-15_2D for all patterns, except Bit Complement, since average hop counts are ≤ 8 for an 8×8 Mesh.

7.1.2. SA-G priorities. We study the effects of priority in Figure 13a for the best possible 1D and 2D SMART designs. While both priority schemes perform identically at very low-loads, Prio=Bypass has a sudden throughput degradation at an injection rate of about 44-48% of network capacity. Intuitively, we would expect Prio=Bypass to be better than Prio=Local as it allows for longer bypass paths, and avoids unnecessary stopping and buffering of flits already in flight. Moreover, it is often the priority scheme used in non-speculative 1-cycle router designs [24, 27] when choosing between a lookahead and a local buffered flit. However, for SMART, where multiple allocations are occurring in the same cycle, it suffers from a unique problem, highlighted in Figure 13b. In this example, Router's R0, R1 and R3 send *SSRs* up to R2, R4 and R5 respectively, in Cycle-1. In a Prio=Local scheme, R0's *SSR* would lose at R1, and R1's *SSR* would lose at R3, leading to the traversals shown in Cycle-2. For Prio=Bypass, R0's *SSR* will win at R1, and the corresponding flit will be able to go all the way to its stop router R2. However, even though R1's *SSR* lost SA-G at its start router, it wins over R3's *SSR* at R3, preventing R3 from sending its own flit. This cascading effect can continue, leading to poor link utilization, and heavy throughput loss. This effect is reflected in the percentage of false negatives (cases where a router was expecting a flit but no flit came) going up to 25-40% in Prio=Bypass, killing its throughput, as opposed to less than 10% in Prio=Local. On the plus side, Prio=Bypass always creates SMART-hops with high *HPC*s, since a flit that starts only stops at its requested stop router, or at the turn router in this priority scheme. This can be seen in Figure 13c where SMART-8_1D_Prio=Bypass achieves an average *HPC* of 3, while SMART-15_2D_Prio=Bypass maintains an *HPC* of 4-5. Prio=Local, on the other hand, forces the achievable *HPC*s to drop to 1 at high loads.

7.1.3. Impact of HPC_{max}. Next we study the impact of HPC_{max} on performance. We plot the average flit latency for BC traffic (which has high across-chip communication) for HPC_{max} from 1 to 12, across 1D and 2D in Figure 14a. SMART-1_1D is identical to the baseline_1-cycle router (as it does not need SA-G). HPC_{max} of 2 itself gives a 1.8X low-load latency reduction, while 4 gives a 3X reduction. These numbers indicate that even with a faster clock, say 2.25GHz, which will drop HPC_{max} to 4, a SMART-like design is a better choice than a 1-cycle router. It should also be noted that as we scale to smaller feature sizes, cores shrink while die sizes remain unchanged, so the same SMART interconnect length will translate to larger HPC_{max}. Adding SMART_2D, and increasing HPC_{max} to 12 pushes low-load latency close to a 2-cycles: an 8.4X reduction over the baseline. This result highlights that a heavily-pipelined higher frequency baseline can only match SMART if it runs at 8.4GHz.

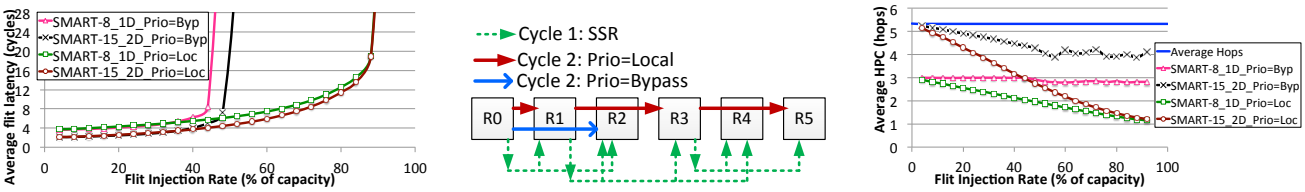
7.1.4. Impact of multi-flit packets. SMART locks an input and output port till all flits of a packet leave for virtual cut-through (Section 4.4). This leads to a poorer switch allocation compared to the baseline which implements flit-by-flit wormhole switching with VCs. Figure 14b evaluates SMART with UR traffic with all packets having 5-flits (a worse case adversarial traffic scenario). We see that SMART achieves its peak throughput with 4-6 VCs, but shows 11% lower throughput than the baseline, even with 12 VCs.

7.1.5. SMART on a 16×16 mesh. Figure 14c plots the performance of SMART on a 256-core mesh with UR traffic. SMART scales well, with HPC_{max}=4 lowering network latency from 23 to 6-7 cycles at low loads. SMART-11_1D and SMART-9_2D lower it even further to 3-4 cycles. SMART also gives a 12% throughput improvement.



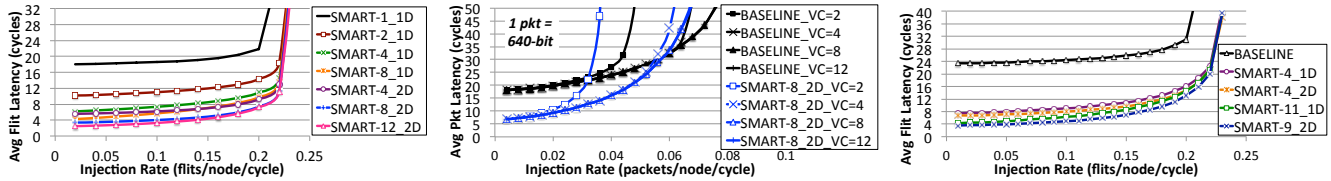
(a) Uniform Random (UR) (Avg Hops = 5.33) (b) Bit Complement (BC) (Avg Hops = 8) (c) Transpose (TP) (Avg Hops = 6)

Figure 12: SMART with synthetic traffic



(a) Average Network Latency (b) Throughput Loss in Prio=Bypass (c) Average achievable HPC

Figure 13: Prio=Local vs Prio=Bypass for Uniform Random Traffic.



(a) Impact of HPC_{max} (Bit Complement) (b) Impact of 5-flit packets (Uniform Random) (c) 256-core (Uniform Random)

Figure 14: Features of SMART

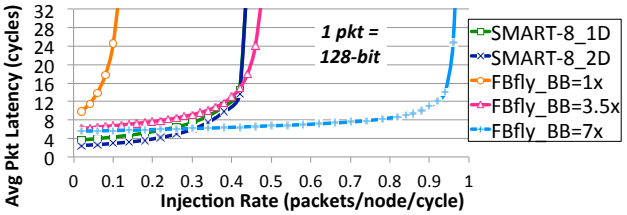


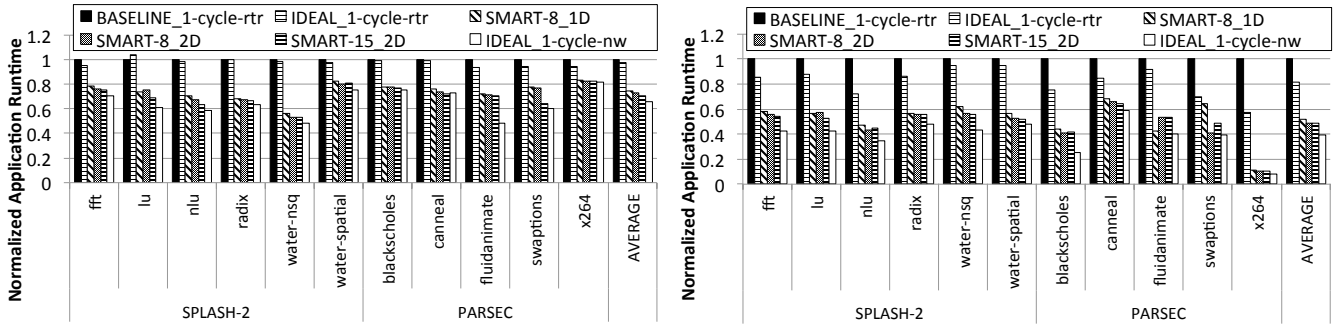
Figure 15: SMART vs Flattened Butterfly (Uniform Random)

7.1.6. Comparison with High-Radix Topology. We compare SMART with a Flattened Butterfly [22] topology. Each FBfly router has dedicated single-cycle links to every other node in that dimension (7 ports per direction + NIC port, i.e. radix-29). We assume that the router delay is 1-cycle. This is a very aggressive assumption, especially because the SA stage needs to perform 22:1 arbitrations. All high-radix routers assume > 4-cycle pipelines [22, 21, 34]. We use 8VCs per port with virtual cut-through in both SMART and FBfly (thus giving more buffer resources to FBfly). In Figure 15, we plot three configurations where the total number of wires, i.e. Bisection Bandwidth (BB), of the FBfly is 1x, 3.5x and 7x that of SMART (leading to 7-flits, 2-flits and 1-flit per packet respectively for 128-bit packets). At BB=1x, FBfly loses both in latency and throughput due to heavy serialization delay. At BB=3.5x, FBfly can match SMART in throughput. Despite an aggressive 1-cycle router, at BB=7x the best case latency for FBfly is 6 cycles (2 at injection, 2 at turning, and 2 at ejection router) as compared to 4 and 2 for SMART-1D and SMART-2D respectively. The radix-29 FBfly_BB=3.5x router, modeled in DSENT [35], incurs an area, dynamic power (at saturation) and leakage power overhead of 8.6x, 1.5x and 10x respectively over SMART. If we are willing to use N times more wires, a better solution would be to just have N meshes, each with SMART, so that fast latency is achieved in addition to scalable bandwidth.

7.2. Full-system Traffic

We evaluate the parallel sections of SPLASH-2 [3] and PARSEC [9] for both Private and Shared L2. Each run consists of 64 threads of the application running on our CMP. We run 5-10 times with different random seeds to capture variability in parallel workloads [6], and average the results.

7.2.1. Performance Impact. Figure 16 shows that SMART-8_1D and SMART-8_2D lower application runtime by 26% and 27% respectively on average, for a Private L2, which is only 8% away from an ideal 1-cycle network. The runtime reduction goes up to 49% and 52% respectively with a Shared L2 design, which is 9% off from an ideal 1-cycle network. SMART-15_2D does not give any significant runtime benefit over SMART-8_2D.



(a) Private L2 (b) Shared L2

Figure 16: Full-system application runtime with SMART

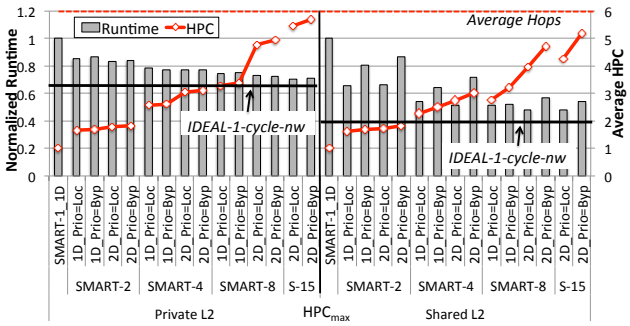


Figure 17: Impact of HPC_{max} and Priority

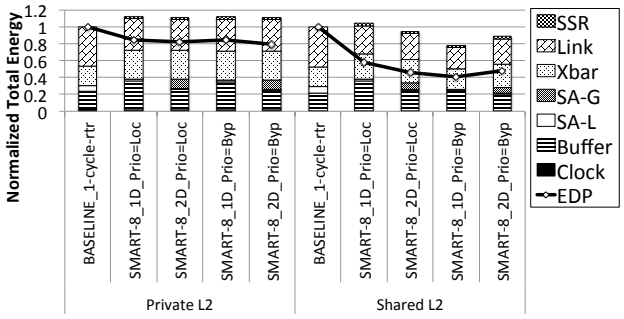


Figure 18: Total Network Dynamic Energy

7.2.2. Impact of HPC_{max} and priority. Figure 17 sweeps through HPC_{max} and SA-G priority, and plots the normalized runtime and achieved HPC, on average across all the benchmarks. Since these full-system traffic fall in the lower end of the injection rates in the synthetic traffic graphs, Prio=Bypass performs almost as well as Prio=Local, except at low HPC_{max} in a Shared L2. HPC_{max} of 4 suffices to achieve most of the runtime savings.

7.2.3. Total Network Energy. Figure 18 explores the energy trade-off of SMART by plotting the total dynamic energy of the network consumed for running the benchmarks to completion, on average across all the benchmarks. For Private L2, the dynamic energy for SMART goes up by 10-12% across designs primarily due to the data-path, though the overall EDP goes down by 20%. For Shared L2, the dynamic energy goes down by 6-21% across the designs, because of a lower runtime. The EDP goes down by up to 59%. SMART with Prio=Local consumes 18-46% higher energy in the buffers than both SMART with Prio=Bypass and the baseline (which also prioritizes incoming flits over already buffered local flits in SA to reduce buffering), since Prio=Bypass, by definition, reduces the number of times flits need to stop and get buffered. SA-G energy contributes less than 1% of network energy for SMART_1D, and goes up to about 10% for SMART_2D. All these ups and downs are however negligible when we also consider leakage. We observed leakage to contribute more than 90% of the total energy, since the network activity is very low in full-system scenarios[16]. However, even with high leakage, the total network power was observed to be about 3W for both baseline and SMART, while chip power budgets are usually about 100W. Thus the energy overheads of SMART are negligible.

8. Related Work

High-radix routers. High-radix router designs such as CMesh [8], Fat Tree [11], Flattened Butterfly [22], BlackWidow [34], MECS [13], Clos [19] are topology solutions to reduce average hop counts, and advocate adding physical express links between distant routers. These express Pt-to-Pt links can be further engineered for lower delay with advanced signaling techniques like equalization [20] and capacitively-driven links [24]. Each router now has > 5 ports, and channel bandwidth (b) is often reduced proportionally to have similar buffer and crossbar area/power as a mesh (radix-5) router. More resources however imply a complicated routing and Switch+VC allocation mechanism, with a hierarchical SA and crossbar [21], increasing router delay t_r to 4-5 at the routers where flits do need to stop. The pipeline optimizations described earlier are hard to implement here. These designs also complicate layout since multiple Pt-to-Pt global wires need to span across the chip. Moreover, a topology solution only works for certain traffic, and incurs higher latencies for adversarial traffic (such as near neighbor) because of higher serialization delay.

[16] Leakage could be reduced by aggressive power gating solutions, which itself is a research challenge.

In contrast, SMART provides the illusion of dedicated physical express channels, embedded within a regular mesh network, without having to lower the channel bandwidth, or increase the number of router ports.

Asynchronous NoCs. Asynchronous NoCs [10, 7] have been proposed for the SoC domain for deterministic traffic. Such a network is programmed statically to preset contention-free routes for QoS, with messages then transmitted across a fully asynchronous NoC (routers and links). Instead, SMART couples *clocked* routers with asynchronous links, so the routers can perform fast cycle-by-cycle reconfiguration of the links, and thus handle general-purpose CMPs with non-deterministic traffic and variable contention scenarios. Asynchronous Bypass Channels [17] target chips with multiple clock domains across a die, where each hop can incur significant synchronization delay. They aim to remove this synchronization delay. This leads them to propose sending a clock signal with the data so that the data can be latched correctly at the destination router. Besides this difference in link architecture, the different goals also lead to distinct NoC architectures. Due to the multiple clock domains, ABC needs to buffer/latch flits at every hop speculatively, discarding them thereafter if flits successfully bypassed. Also, the switching between *bypass* and *buffer* modes cannot be done cycle-by-cycle, which increases latency. In contrast, SMART targets a single clock domain across the entire die, so SSRs can be sent in advance to avoid latching flits at all along a multi-hop path and allow routers to switch between *bypass* and *buffer* modes each cycle.

9. Conclusion

Aggressive NoC pipeline optimizations have been able to lower router delays to just 1-cycle. However, this is not good enough for large networks with multi-hop paths. The solution of adding explicit fast physical channels to bypass routers comes with its own set of problems in terms of layout complexity, area and power. We present SMART, a solution to traverse multi-hop paths within a single-cycle, by virtually bypassing all routers along the route, without adding any physical channels on the data-path. This work opens up a plethora of research opportunities in circuits, NoC architectures and many-core architectures to optimize and leverage SMART NoCs. We see SMART paving the way for locality-oblivious CMPs, easing the burden on coherence protocol and/or software from optimizing for locality.

References

[1] `http://www.tilera.com/products/processors/TILE-Gx_Family`.

[2] `http://projects.csail.mit.edu/angstrom`.

[3] `http://www-flash.stanford.edu/apps/SPLASH/`.

[4] `http://www.windriver.com/products/simics`.

[5] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *ISPASS*, pages 33–42, 2009.

[6] A. R. Alameldeen and D. A. Wood. Variability in architectural simulations of multi-threaded workloads. In *HPCA*, 2003.

[7] J. Bainbridge and S. Furber. Chain: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, Sept 2002.

[8] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *ICS*, pages 187–198, 2006.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, 2008.

[10] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *DATE*, pages 1226–1231, 2005.

[11] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Pub., 2003.

[12] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees. In *ISCA*, pages 401–412, 2011.

[13] B. Grot *et al.* Express cube topologies for on-chip interconnects. In *HPCA*, pages 163–174, 2009.

[14] R. Haring *et al.* The IBM Blue Gene/Q Compute Chip. *IEEE Micro*, 32(2):48–60, Mar 2012.

[15] Y. Hoskote *et al.* A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, Sept. 2007.

[16] J. Howard *et al.* A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *ISSCC*, pages 108–109, 2010.

[17] T. N. K. Jain, P. V. Gratz, A. Sprintson, and G. Choi. Asynchronous bypass channels: Improving performance for multi-synchronous nocs. pages 51–58, 2010.

[18] D. Johnson *et al.* Rigel: A 1,024-core single-chip accelerator architecture. *IEEE Micro*, 31(4):30–41, Jul 2011.

[19] Y.-H. Kao *et al.* CNoC: High-radix clos network-on-chip. *TCAD*, 30(12):1897–1910, 2011.

[20] B. Kim and V. Stojanović. Equalized interconnects for on-chip networks: modeling and optimization framework. In *ICCAD*, pages 552–559, 2007.

[21] J. Kim *et al.* Microarchitecture of a high-radix router. In *ISCA*, pages 420–431, 2005.

[22] J. Kim *et al.* Flattened butterfly topology for on-chip networks. In *MICRO*, pages 172–182, 2007.

[23] T. Krishna, L.-S. Peh, B. M. Beckmann, and S. K. Reinhardt. Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *MICRO*, pages 71–82, 2011.

[24] T. Krishna *et al.* Express Virtual Channels with capacitively driven global links. *IEEE Micro*, 29(4):48–61, 2009.

[25] T. Krishna *et al.* SWIFT: A SWing-reduced Interconnect For a Token-based network-on-chip in 90nm CMOS. In *ICCD*, pages 439–446, 2010.

[26] A. Kumar *et al.* A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *ICCD*, pages 63–70, 2007.

[27] A. Kumar *et al.* Token Flow Control. In *MICRO*, 2008.

[28] G. Kurian *et al.* ATAC: a 1000-core cache-coherent processor with on-chip optical network. In *PACT*, pages 477–488, 2010.

[29] M. M. K. Martin *et al.* Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *CAN*, 2005.

[30] H. Matsutani *et al.* Prediction router: Yet another low latency on-chip router architecture. In *MICRO*, pages 367–378, 2009.

[31] R. Mullins *et al.* Low-latency virtual-channel routers for on-chip networks. In *ISCA*, pages 188–197, 2004.

[32] S. Park *et al.* Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI. In *DAC*, pages 398–405, 2012.

[33] J. Rabaey and A. Chandrakasan. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall Pub., 2002.

[34] S. Scott *et al.* The BlackWidow high-radix clos network. In *ISCA*, pages 16–28, 2006.

[35] C. Sun *et al.* DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *NOCS*, pages 201–210, 2012.

[36] D. Wentzlaff *et al.* On-chip interconnection architecture of the Tile Processor. *IEEE Micro*, 27(5):15–31, Sept. 2007.