SMART: SINGLE-CYCLE MULTIHOP TRAVERSALS OVER A SHARED NETWORK ON CHIP

SMART (SINGLE-CYCLE MULTIHOP ASYNCHRONOUS REPEATED TRAVERSAL) AIMS TO DYNAMICALLY SET UP SINGLE-CYCLE PATHS (WITH TURNS) FROM THE SOURCE TO THE DESTINATION FOR MESSAGE FLOWS SHARING A NETWORK ON CHIP. A FLOW-CONTROL TECHNIQUE ARBITRATES FOR AND RESERVES MULTIPLE LINKS WITHIN A CYCLE. A ROUTER AND LINK MICROARCHITECTURE ENABLES A MULTIHOP (9 TO 11 HOPS AT 1 GHZ IN 45 NM) TRAVERSAL WITHIN A CYCLE.

Tushar Krishna
Chia-Hsin Owen Chen
Woo-Cheol Kwon
Li-Shiuan Peh
Massachusetts Institute of Technology

Increasing the number of on-chip cores continues to be the de facto strategy to scale performance in the presence of two trends: technology scaling (that is, more transistors) due to Moore's law, and single-core frequency plateauing due to the Power Wall. These cores are often connected by a shared interconnect fabric, such as a ring or a mesh, over which multiple communication flows multiplex. As core counts go up, the average number of "hops" between communicating cores goes up as well—linearly with k for a k-node ring or a $k \times k$ mesh. (We define a *hop* to be the physical distance between neighboring tiles. In this paper, 1 hop = 1 mm, based on place-and-route of a Freescale PowerPC e200z7 core in 45 nm.)

The number of hops directly impacts the communication latency T_N for a *flit*, which is the smallest unit of a network packet, and the granularity at which network resources (links and buffers) are allocated. This can be expressed as:

$$T_N = H(t_r + t_w) + \sum_{h=1}^{H} t_c(h), \qquad (1)$$

where H is the number of hops, t_r is the router's intrinsic delay, t_w is the wire (between two routers) delay, and $T_c = \sum t_c(h)$ is the network contention delay—the number of cycles spent waiting to get access to the switch and output link. Multiflit packets incur an additional component T_s or a serialization delay, which is set by the number of cycles that a packet of length L takes to cross a channel with bandwidth b (that is, the number of flits in the packet). The on-chip latency in turn affects the completion time of cache coherence transactions, because cache misses often must be serviced by remote caches or memory controllers. Slower requests and responses lead to a slower injection of new requests due to dependencies, which leads to poorer throughput and overall system slowdown. The increased on-chip latency due to the increased number of hops is a worrisome trend, especially with ambitious design goals of adding hundreds of cores to a single chip for the exascale era.

In this work, we present a solution to achieve close to an ideal one-cycle network ($T_N = 1$) traversal on a mesh for any source-destination pair. Our proposed network on a chip (NoC) is called Smart (Single-cycle Multihop 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 (11 in this work) within a single clock cycle before getting latched. We present a network flow-control mechanism to set up arbitrary multihop paths with turns (that is, repeated wires on demand) within a cycle, and then traverse them within a cycle. We optimize network latency as follows:

$$T_N = \lceil (H/HPC) \rceil \cdot (t_r + t_w) + \sum_{h=1}^{H} t_c(h), \qquad (2)$$

where *HPC* stands for number of hops per cycle, the maximum value for which ($HPC_{\max}$) depends on the underlying technology. We reduce the effective number of hops to $\lceil (H/HPC) \rceil$, without adding any additional physical wires in the datapath between distant nodes.

On a 64-core mesh, synthetic traffic shows a 5- to 8-times reduction in average network latency, whereas full-system Splash-2 and Parsec traffic shows a 27 and 52 percent reduction in average runtime for private and shared level-2 (L2) designs, respectively, compared to a state-of-the-art NoC with one-cycle routers. Smart is a more scalable and less expensive solution than alternate approaches to reduce network latency, which are discussed in the "High-Radix Routers and Asynchronous NoCs" sidebar.

Background

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-and-link traversal. A router performs the following actions:[1]

- *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 Virtual Channel (VC) for the next router, from a pool of free VCs.[2]
- The winners of SA proceed to *Switch (crossbar) Traversal* (*ST*) and *Link Traversal* (*LT*) to reach the next routers.

A 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 look-ahead routing,[1] simplified VC selection,[2] speculative switch arbitration,[3,4] nonspeculative switch arbitration via look-aheads[5,6] to bypass buffering, and so on. This has allowed the router delay t_r (Equation 1) to drop from three to five cycles in industry prototypes[7,8] to one cycle in academic NoC-only prototypes.[6] We use this state-of-the-art one-cycle router as our baseline. ST and LT can be done together within a cycle,[6,8] giving us $t_w = 1$. Thus, our baseline incurs two cycles per hop (see Figure 1). In case of contention, flits have to be buffered and could wait multiple cycles before they win SA and VS, increasing T_c, as shown at Router$_{n+\mathrm{i}}$.

The Smart interconnect

Adding asynchronous repeaters (that is, a pair of inverters) at regular intervals on a long wire is a standard way to reduce wire delay.[9,10] We perform a design-space exploration of repeated wires in a commercial 45-nm silicon on insulator (SOI) technology using the place-and-route tool Cadence

High-Radix Routers and Asynchronous NoCs

High-radix router designs such as Fat Tree,[1] Flattened Butterfly,[2] and Clos[3] are topology solutions to reduce average hop counts. They advocate adding physical express links between distant routers. Each router now has more than five ports, and channel bandwidth (*b*) is often reduced proportionally to have similar buffer and crossbar area and power as a mesh (radix-5) router, increasing the total number of flits, adding serialization delay T_s to each packet. Moreover, more ports complicates the routing, switch allocation, and virtual channel allocation mechanism, often requiring a hierarchical switch allocator and crossbar,[4] increasing router delay t_r to 4 to 5 at the routers where flits must stop. The pipeline optimizations described in the main article are hard to implement here. These designs also complicate layout because multiple point-to-point global wires must span the chip. Moreover, a topology solution works only for certain traffic, and incurs higher latencies for adversarial traffic (such as near neighbor) because of higher T_s. 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. Given the same number of wires, it can virtually create the same high-radix topology with lower t_r and no additional T_s.

Asynchronous networks on chip (NoCs) have been proposed for the system-on-a-chip (SoC) domain for deterministic traffic.[5] Such a network is programmed statically to preset contention-free routes for quality of service, 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 chip multiprocessors with nondeterministic traffic and variable contention scenarios. Asynchronous bypass channels target chips with multiple clock domains across a die,[6] 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. However, unlike Smart, bypass and buffer modes cannot be switched cycle by cycle, and flits must be speculatively latched at every hop.

References

1. W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2003.
2. J. Kim et al., "Flattened Butterfly Topology for On-Chip Networks," *Proc. 40th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2007, pp. 172-182.
3. Y.-H. Kao et al., "CNoC: High-Radix Clos Network-on-Chip," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 12, 2011, pp. 1897-1910.
4. J. Kim et al., "Microarchitecture of a High-Radix Router," *Proc. 32nd Ann. Int'l Symp. Computer Architecture* (ISCA 05), 2005, pp. 420-431.
5. T. Bjerregaard and J. Sparso, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip," *Proc. Conf. Design, Automation and Test in Europe* (DATE 05), 2005, pp. 1226-1231.
6. T.N.K. Jain et al., "Asynchronous Bypass Channels: Improving Performance for Multisynchronous NoCs," *Proc. 4th ACM/IEEE Int'l Symp. Networks-on-Chip* (NOCS 10), 2010, pp. 51-58.

Encounter. We fix the repeater spacing to 1 mm (our tile size), wire spacing to 3 times the minimum allowed by the technology (to lower the coupling capacitance), and keep increasing the length of the wire, letting the tool size the repeaters appropriately, until it fails timing closure at our target cycle time of 1 nanosecond (ns)—that is, 1 GHz. We translate the maximum distance that a signal can be transmitted within 1 ns into a network microarchitectural parameter hops per cycle max ($HPC_{\max}$):

$$HPC_{\max} = (\text{maximum mm per ns} \times \text{clock period in ns}) / (\text{tile width in mm})$$

Figure 2 shows that $HPC_{\max}$ for repeated wires at 45 nm is 16 (assuming 1 mm tiles and 1 GHz clock). (The place-and-route tool was found to zigzag wires to fit a fixed global grid, adding unnecessary wire length, limiting $HPC_{\max}$. A custom design can potentially go further and with a flatter energy profile, as projected by the timing-driven NoC power-modeling tool Dsent[11]). We observe a similar trend for $HPC_{\max}$ at 32 nm and 22 nm, with energy going down by 19 and 42 percent. At smaller technology nodes, global wires are not expected to become faster (unlike transistors); however, smaller tile sizes and fairly constant frequencies should translate to a higher $HPC_{\max}$.

Router logic delay limits the network frequency to 1 to 2 GHz at 45 nm.[6,8] Link drivers are accordingly sized to drive only 1 mm (1 hop) in 0.5 to 1 ns, before the signal is latched at the next router. Smart removes this

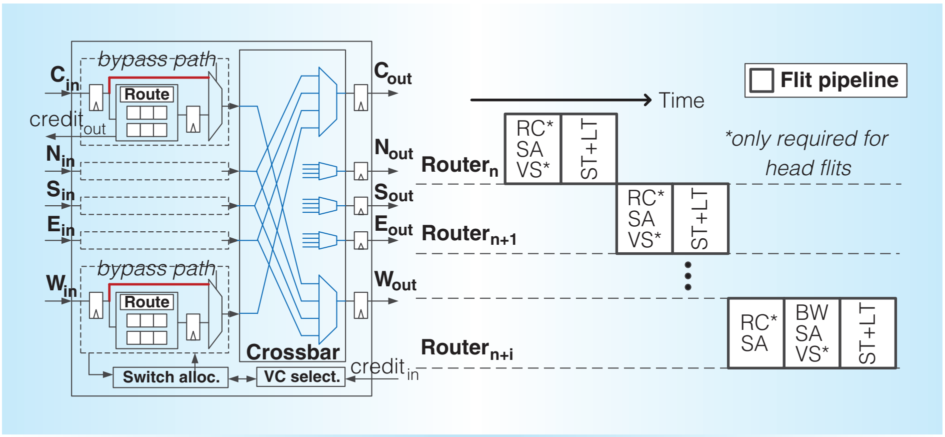

Figure 1. Microarchitecture and pipeline of a state-of-the-art baseline ($t_r = 1$) router. Each network traversal takes two cycles per hop ($t_r = RC + SA + VS$, $t_w = ST + LT$).

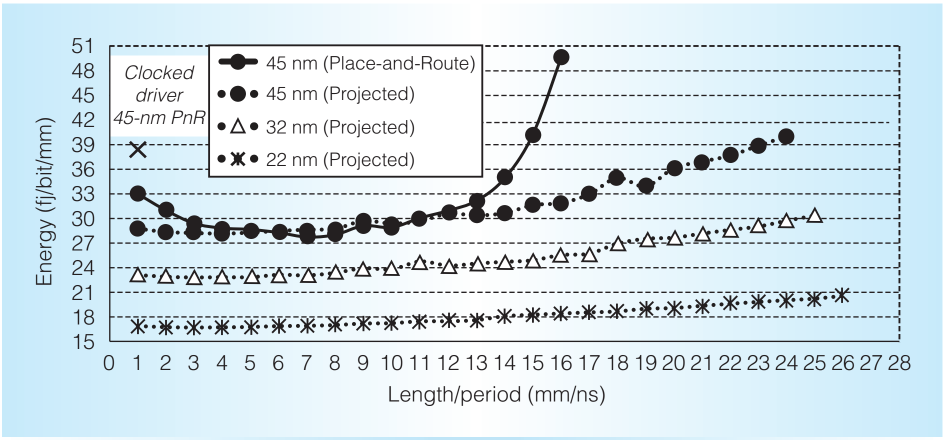

Figure 2. Transmission energy as a function of transmission distance for repeated links at 1 GHz. A placed-and-routed repeated wire in 45 nm can go up to 16 nm in 1 ns. (Wire width: $DRC_{\min}$; wire spacing: $3 \cdot DRC_{\min}$; metal layer: M6; repeater spacing: 1 mm.)

constraint of latching signals at every hop. We exploit the positive slack in the link-traversal stage by replacing clocked link drivers with 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 2. If we choose a 2-mm tile size, or a 2-GHz frequency, $HPC_{\max}$ will go down by half. Asynchronous repeaters also consume 14.3 percent lower energy per bit per mm than conventional clocked drivers, as Figure 2 shows, 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 (such as Intel's 80-core 5-GHz five-stage router[7]), which, even if it were possible to do, does not reduce traversal latency (only improves throughput) and adds huge power overheads owing to pipeline registers.

Figure 3a shows a Smart router. For simplicity, we only show $Core_{in}$ (C_{in}), $West_{in}$ (W_{in}), and $East_{out}$ (E_{out}) ports. (C_{in} does not have a bypass path like the other ports because all flits from the network interface controller [NIC] must be buffered at the first router before they can create Smart paths.) All other input ports are identical to W_{in}, and all other output ports are identical to E_{out}. Each repeater must be sized to drive not just the link, but also the muxes (2:1 bypass and 4:1 crossbar) at the next router, before a new repeater is encountered. Using the same methodology with Cadence

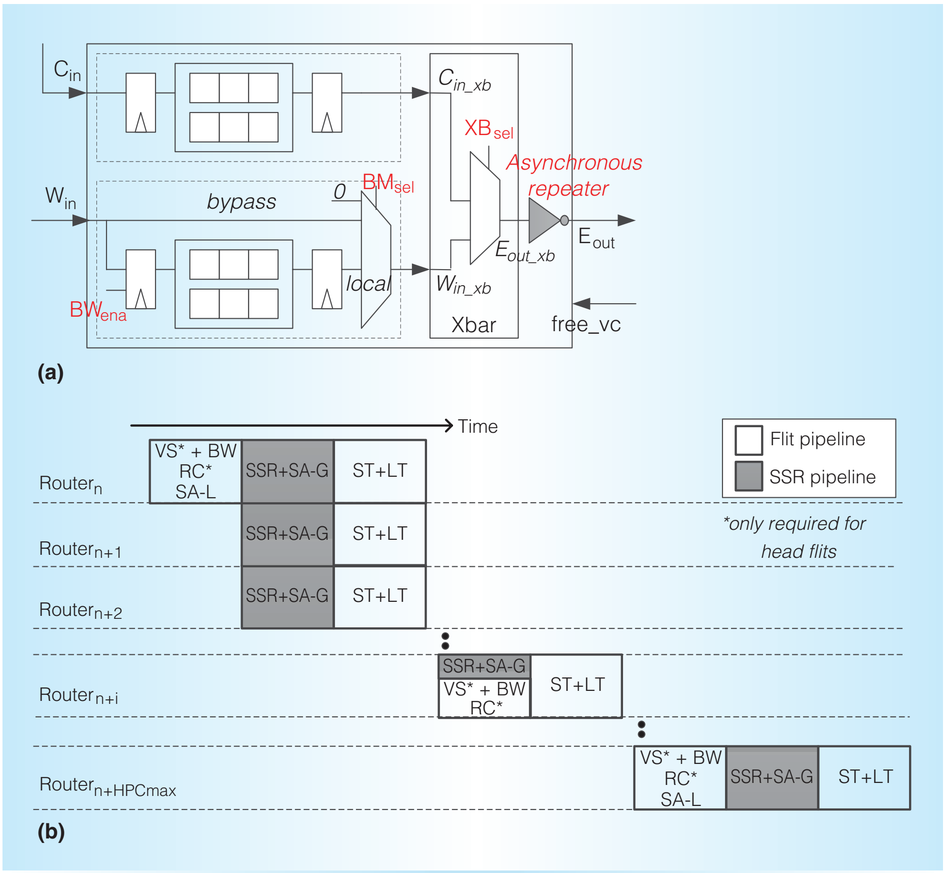

Figure 3. Changes to the router and pipeline to support single-cycle multihop traversals. Smart router microarchitecture (a) and pipeline (b). BW_{ena}, BM_{sel}, and XB_{sel} are set up during the control path (SSR+SA-G). During the datapath (ST+LT), the flit can cross multiple routers in a cycle if BW_{ena} is 0, and BM_{sel} is set to bypass, and gets latched at the router where BW_{ena} is 1.

Encounter, this reduces $HPC_{\max}$ to 11 at 1 GHz.

Figure 3a shows the three primary components of the design:

- Buffer Write enable (BW_{ena}) at the input flip-flop, which determines whether the input signal is latched;
- Bypass Mux select (BM_{sel}) at the input of the crossbar, which chooses between the local buffered flit and the bypassing flit on the link; and
- Crossbar select (XB_{sel}).

In the next section, we describe the flow control to preset these signals.

Smart in a *k*-ary 1-mesh

We start by demonstrating how Smart works in a *k*-ary 1-mesh (see Figure 4). Each router has three ports: West, East, and Core. (For illustration purposes, we only show C_{in}, W_{in}, and E_{out} in the figures.) As Figure 3a shows, E_{out_xb} can be connected either to C_{in_xb} or W_{in_xb}. The latter can be driven either by bypass, local, or 0, depending on BM_{sel}.

The design is called *Smart_1D* (because routers can be bypassed only along one dimension). Bypassing routers at turns in a *k*-ary 2-mesh will be described later. For purposes of illustration, we will assume $HPC_{\max}$ to be 3.

Smart-hop setup request

Figure 3b shows the Smart router pipeline. A *Smart-hop* (single-cycle multihop path) begins from a *start router* (where flits are buffered). Unlike the baseline router,

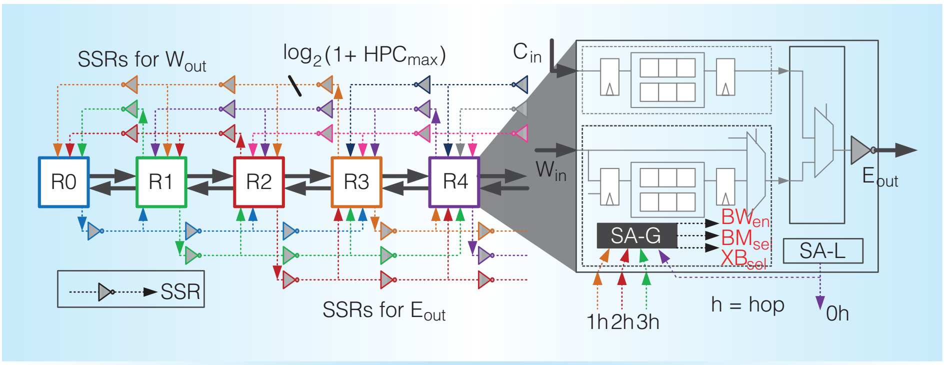

Figure 4. *k*-ary 1-mesh with dedicated Smart-hop setup request (SSR) links going up to HPCmax (3 in this example) hops in each direction. The switch allocation local (SA-L) grant sends SSRs. The switch allocation global (SA-G) unit sets BW_{ena}, BM_{sel}, and XB_{sel} based on the SSRs from 0-hop, 1-hop, 2-hop, and 3-hop neighbors.

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): every start router chooses a winner for each output port from among its buffered (local) flits. In the next cycle, each output port winner first broadcasts a Smart-hop setup request (SSR) up to $HPC_{\max}$-hops from that output port. These SSRs are dedicated repeated wires (which are inherently multidrop) on the control path that connect every router to a neighborhood of up to $HPC_{\max}$ (see Figure 4). SSRs are $log_2(1+HPC_{\max})$-bits wide, and carry the length (in hops) up to which the winning flit wishes to go. For instance, SSR = 2 indicates a 2-hop path request. Each flit tries to go as close as possible to its destination router; hence, SSR = $\min(HPC_{\max}, H_{remaining})$.

During SA-G, all intermediate 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 I/O 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 up to multiple hops till they are stopped by BW_{ena} at some router. Thus, flits spend at least two cycles (SA-L and SA-G) at a start router before they can use the switch. SSR traversal and SA-G occur serially within the same cycle.

Single-cycle multihop paths are opportunistic, not guaranteed; flits can end up getting prematurely stopped (that is, before their SSR length) depending on the SA-G results at different routers, which depends on contention.

We illustrate all this with examples. In Figure 5, 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 $SSR_D = 2$ (that is, a 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 and $XB_{sel} = W_{in\_xb} \rightarrow E_{out\_xb}$. At R3, which is 1 hop away ($< SSR_D$), BM_{sel} = bypass and $XB_{sel} = W_{in\_xb} \rightarrow E_{out\_xb}$. 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 6. In Cycle 1, R2 sends out SSR_D as before; 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. For example, the *Prio=Local* scheme gives highest priority to the local (buffered) flit, followed by the flit from the neighboring router, followed by the flit from the router two hops away, and so on; so $Flit_E$ loses to $Flit_D$. Figure 6 shows the

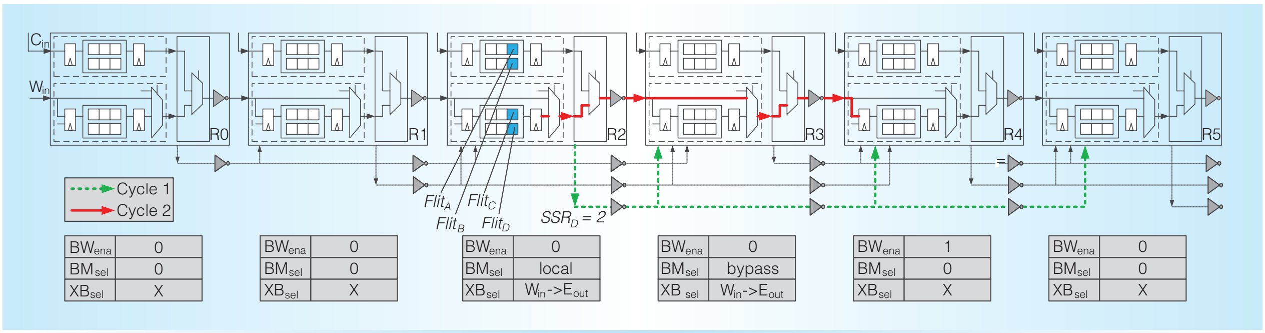
C_{in}
W_{in}
R0
R1
R2
R3
R4
R5
Cycle 1
Cycle 2
$Flit_A$
$Flit_B$
$Flit_C$
$Flit_D$
$SSR_D = 2$
BW_{ena} 0, BM_{sel} 0, XB_{sel} X
BW_{ena} 0, BM_{sel} 0, XB_{sel} X
BW_{ena} 0, BM_{sel} local, XB_{sel} W_{in}->E_{out}
BW_{ena} 0, BM_{sel} bypass, XB_{sel} W_{in}->E_{out}
BW_{ena} 1, BM_{sel} 0, XB_{sel} X
BW_{ena} 0, BM_{sel} 0, XB_{sel} X

Figure 5. Smart example: no SSR conflict. Cycle 1: $Flit_D$ (assumed to have won SA-L in Cycle 0) sends $SSR_D = 2$; that is, a request to bypass R3 and stop at R4. The SA-G units at R2, R3, and R4 set up a single-cycle multihop bypass path. Cycle 2: $Flit_D$ starts at R2 and gets latched at R4.

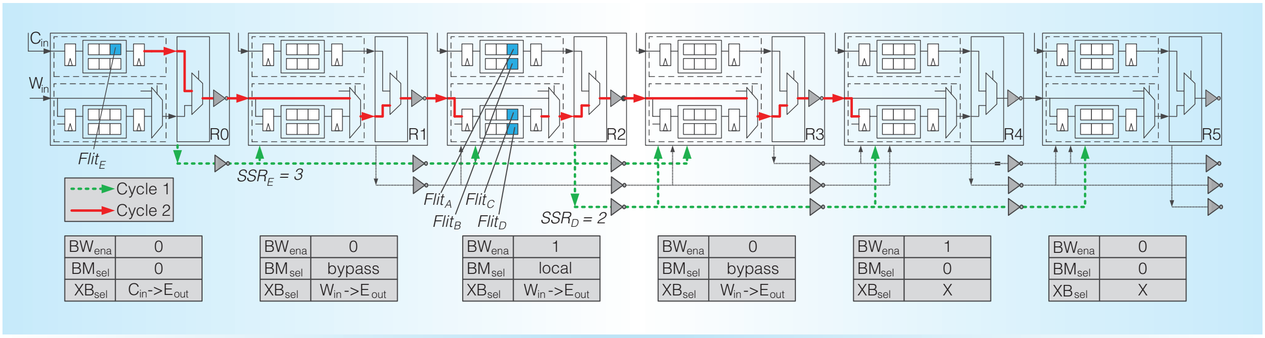
C_{in}
W_{in}
R0
R1
R2
R3
R4
R5
$Flit_E$
$SSR_E = 3$
Cycle 1
Cycle 2
$Flit_A$
$Flit_B$
$Flit_C$
$Flit_D$
$SSR_D = 2$
BW_{ena} 0, BM_{sel} 0, XB_{sel} C_{in}->E_{out}
BW_{ena} 0, BM_{sel} bypass, XB_{sel} W_{in}->E_{out}
BW_{ena} 1, BM_{sel} local, XB_{sel} W_{in}->E_{out}
BW_{ena} 0, BM_{sel} bypass, XB_{sel} W_{in}->E_{out}
BW_{ena} 1, BM_{sel} 0, XB_{sel} X
BW_{ena} 0, BM_{sel} 0, XB_{sel} X

Figure 6. Smart example: SSR conflict with Prio=Local. The Prio=Local scheme gives highest priority to the local (buffered) flit, then the flit from the neighboring router, followed by the flit from the router two hops away, and so on. $Flit_E$ from R0 is prematurely stopped at R2, before its intended destination R3, to allow R2 to send its own local $Flit_D$ on its East output link.

values of BW_{ena}, BM_{sel}, and XB_{sel} at each router for this priority. 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 with an empty buffer slot is guaranteed to exist whenever a flit is made to stop, as we will explain later. An alternate priority, *Prio=Bypass,* prioritizes flits from the furthest router over the flits from the nearer ones. Here, in Cycle 2, $Flit_E$ would traverse all the way from R0 to R3, and $Flit_D$ would be stalled.

False positives and false negatives

Can a flit arrive at a router, even though the router isn't expecting it (that is, a false positive)? The answer is no. For correctness, all routers must enforce the same SA-G priority (Prio=Local or Prio=Bypass), thus ensuring the same relative priority between the SSRs. All flits that arrive at a router are expected and will stop or bypass on the basis of their SSR's success in the previous cycle. Different routers choosing different SA-G priorities could result in misrouting beyond the allowed HPC_{max}-hops.

Can a flit not arrive at a router, even though the router is expecting it (that is, a false negative)? Yes. It is possible for the router to be set up for stop or bypass for some flit, but no flit arrives. This can happen if that flit was forced to prematurely stop at some prior router, owing to some SSR interaction at that router 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 (the valid-bit from the flit is thus used in addition to BW_{ena} when deciding whether to buffer). Unlike false positives, this is not a correctness issue but rather a performance (throughput) issue, because some links go idle, when they could have been used by other flits if more global information were available.

Ordering

In Smart, any flit can be prematurely stopped on the basis of the interaction of SSRs that cycle. We must ensure that this does not result in reordering between flits of the same packet, or between flits from the same source (if point-to-point ordering is required in the coherence protocol).

The first constraint is in routing (relevant to 2D topologies). Multiflit packets and point-to-point ordered virtual networks should use only 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 an SSR is received at an input port, and there is either a prematurely buffered head/body flit or a prematurely buffered flit within a point-to-point ordered virtual network, the incoming flit is stopped.

Guaranteeing free VC with 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 VC ID for its next router (neighbor) before it leaves the router. The neighbor signals back when that VC ID 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 VC ID before it leaves will not work because it is not guaranteed to reach its presumed next router, and multiple flits at different start routers might end up choosing the same VC ID. 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. (If the router has multiple virtual networks, or *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 intermediate routers will know which vnet's free VC signal to look at.)

During SA-G, if an SSR requests an output port without a free VC, BW_{ena} is made high and the corresponding flit is buffered. This solution does not add any extra multihop 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; otherwise, the flit is stopped at the previous router.

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.

Body/tail flits identify which VC to go to at the stop router by using their injection_router ID. Every input port maintains a table to map a VC ID to an injection router ID (the table size equals the number of multiflit VCs at that input port). 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 (that is, virtual cutthrough) to ensure that the body/tail will always have a free buffer in its VC.

What if two body/tail flits with the 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 a router's output port before flits from another packet can leave from that output port. This guarantees a unique mapping from injection_router ID to VC ID in the table at every router's input port.

What if a head bypasses, but the body/tail is prematurely stopped? The body/tail still must identify a VC ID 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 of its intermediate routers, even though it does not stop there. This is done from the bypassing flit's valid, type, and injection_router fields. The tail flit frees the VCs at all the intermediate routers. Thus, for multiflit 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, reducing the buffer turnaround time.

Additional optimizations

We optimize Smart further to push it toward the ideal ($T_N = 1$) NoC.

Bypassing the destination router. So far, we have assumed that a flit starting at an injection router traverses one (or more) Smart-hops until it reaches the destination router, where it gets buffered and requests for the C_{out} port. We add an extra ejection-bit in the SSR to indicate whether the requested stop router corresponds to the destination router for the packet, and not any intermediate router on the route. If a router receives an SSR from H-hops away with value H (that is, a 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.

Bypassing SA-L at low load. We add no-load bypassing[1] 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 3b for Router$_{n+i}$.

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 two cycles in the entire Smart network in the best case (one cycle for SSR and one for ST+LT all the way to the destination NIC).

Smart in a *k*-ary 2-mesh

We demonstrate how Smart works in a *k*-ary 2-mesh. Each router has five ports: West, East, North, South, and Core.

Bypassing routers along dimension

We start with a design in which we do not allow bypass at turns—that is, all flits must stop at their turn routers. We reuse 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 our earlier description. Each output port has multidrop SSR wires spanning upto $HPC_{\max}$-routers along that dimension. Each input port of a router receives $HPC_{\max}$ set of SSR wires, one from each router. 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.

Bypassing routers at turns

In a *k*-ary 2-mesh topology, all routers within an $HPC_{\max}$ neighborhood can be reached within a cycle, as shown in Figure 7a by the shaded diamond. We now describe Smart_2D, which lets flits bypass both the routers along a dimension and the turn routers. We add dedicated SSR links for each possible XY/YX path from every router to its $HPC_{\max}$ neighbors. Figure 7a shows that the E_{out} port has five 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.

In the Smart_2D design, there can be more than one SSR from H-hops away, as shown in the example in Figure 7b for router R_j; it receives SSRs from routers R_m and R_n, which are both one hop away. Router R_k receives the same SSRs. R_j and R_k both need to prioritize the same SSRs to not create false positives (for example, if R_j prioritizes the SSR from R_n and R_k prioritizes the SSR from R_m, the flit from R_n will get misrouted). To arbitrate between SSRs from routers that are the same distance away, we add a second level of priority based on direction. We arbitrarily choose straight-hops > left-hops > right-hops, where straight, left, and right are relative to the I/O port. Figures 7c and 7d plot contours through routers that are the same number of hops away and highlight each router's relative priority. For the intermediate router R_j in Figure 7b, 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 7c. Similarly, at R_k, the SSR from R_m will have higher priority (2_0) over the one

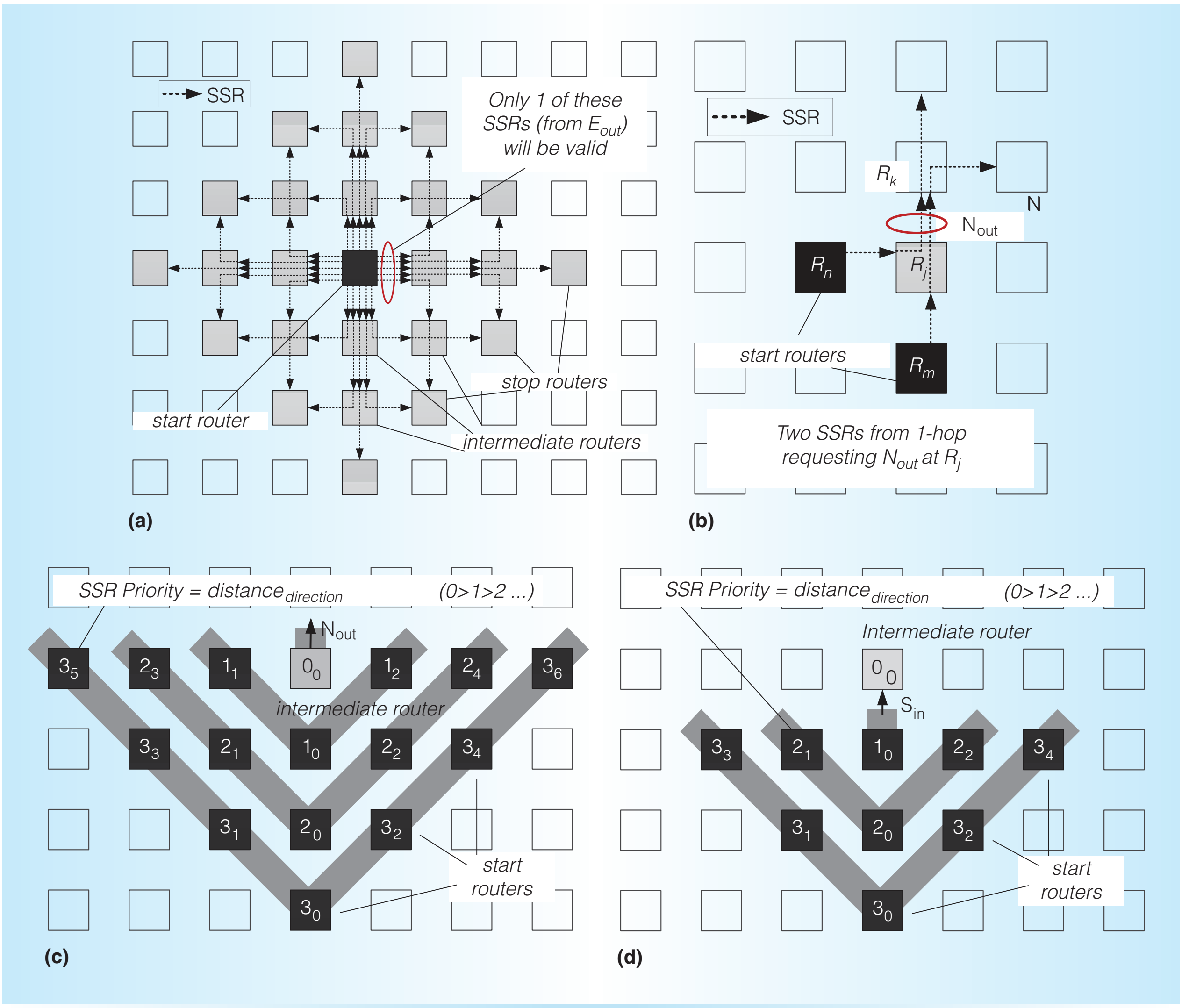

Figure 7. Smart_2D: SSRs and their SA-G priorities. *k*-ary 2-mesh with SSR wires from shaded start router (a). Conflict between two SSRs for N_{out} port (b). Fixed priority at N_{out} port of inter-router (c). Fixed priority at S_{in} port of inter-router (d).

from R_n (2_1) for the S_{in} port, based on Figure 7d. Thus, both routers R_j and R_k will unambiguously prioritize the flit from R_m to use the links, whereas the flit from R_n will stop at Router R_j. We can also infer from Figures 7c and 7d that every router sees the same relative priority for SSRs based on distance and direction, thus guaranteeing no false positives.

Smart implementation

The Smart control path (see Figure 4) consists of $HPC_{\max}$-hops repeated wire delay (SSR traversal), followed by logic gate delay (SA-G). This gave an $HPC_{\max}$ of 13 for Smart_1D and 9 for Smart_2D, following the methodology described earlier. The Smart datapath (see Figure 5) 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 1-mm link. This gave an $HPC_{\max}$ of 11. Picking the lowest of the two gave us an $HPC_{\max}$ of 11 for Smart_1D and 9 for Smart_2D. In our evaluations, we set $HPC_{\max} = 8$, which allows bypass of all routers along the dimension and the ejection, in our target 8-ary 2-mesh.

Evaluation

We use the GEMS[12] and Garnet[13] infrastructure for all our evaluations, which provides a cycle-accurate timing model. All evaluations are for an 8 × 8 mesh. We assume 1 GHz frequency and 45 nm technology. The baseline ($t_r = 1$) in all our runs is a state-of-the-art NoC with one-cycle routers. We also model ideal ($T_N = 1$); this is an ideal but impractical fully connected NoC, in which every flit is magically sent from the source NIC to its destination NIC in one cycle with zero contention. All Smart designs are called Smart-$HPC_{\max}$_1D/2D, and Prio=Local is assumed.

Synthetic traffic

We start by running Smart with synthetic traffic patterns. 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. 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.

Smart across different traffic patterns. Figure 8 compares the performance of three Smart designs: Smart-8_1D and Smart-8_2D (which are both achievable designs), and Smart-15_2D, which reflects the best that Smart can do in an 8 × 8 mesh (with maximum possible hops = 15), against the baseline and ideal. The striking feature about Smart from is that it pushes low-load latency to four and two cycles, for Smart_1D and Smart_2D, respectively, across all traffic patterns, unlike the baseline, for which low-load latency is a function of the average hops. Thus, Smart truly breaks the locality barrier. Smart-8_2D achieves most of the benefit of Smart-15_2D for all patterns, except Bit Complement (BC), since average hop counts are ≤ 8 for an 8 × 8 mesh.

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 9. Smart-1_1D is identical to the baseline ($t_r = 1$) network (as it does not need SA-G). We make two key observations. First, at an $HPC_{\max}$ of 8, Smart shows a 5.4 times reduction in latency. This means that a 1-GHz Smart NoC can be beaten by an NoC with one-cycle routers only

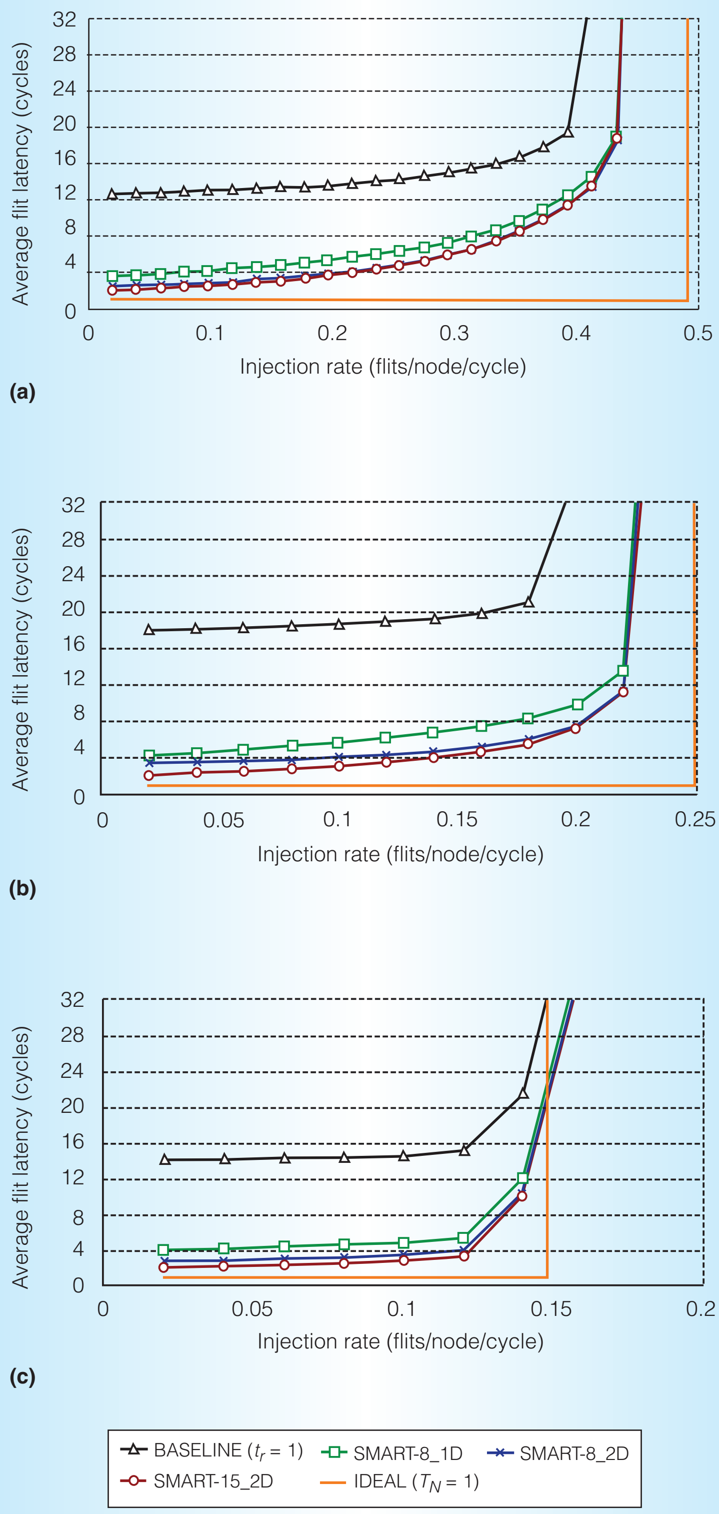

Figure 8. Smart with synthetic traffic. Uniform random (average hops = 5.33) (a). Bit complement (average hops = 8) (b). Transpose (average hops = 6) (c).

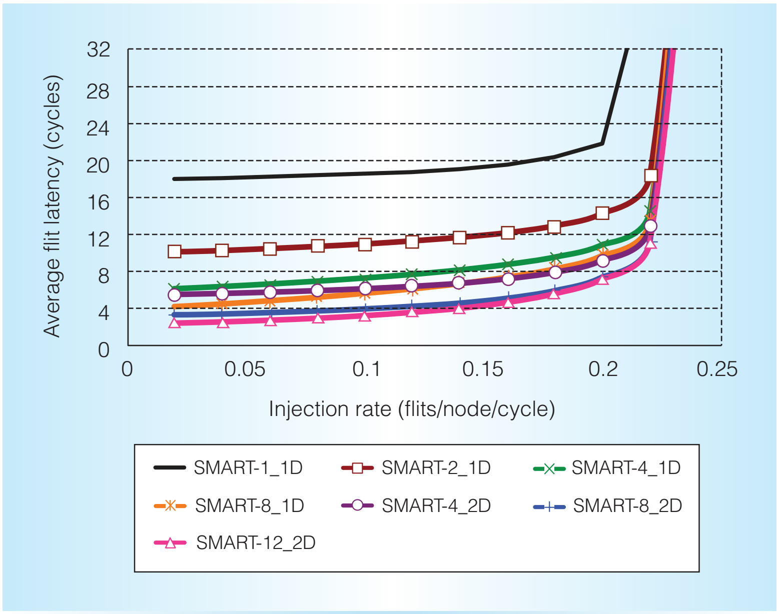

Figure 9. Impact of HPC_{max} with Bit Complement traffic. HPC_{max} of 2 and 4 gives a 1.8 to 3 times reduction in low-load latency compared to HPC_{max} of 1 (that is, baseline ($t_r = 1$)).

if the latter can run at a speed greater than 5.4 GHz, making a strong case for Smart from both a performance and power perspective. A high HPC_{max} would be available in many-core chips with small low-frequency cores for the low-power embedded domain. Second, at a low HPC_{max} of 2 and 4, Smart gives a 1.8 to 3 times reduction in low-load latency compared to HPC_{max} of 1. This makes a Smart-like design a better choice than an NoC with one-cycle routers even in multicores with large high-frequency cores that will limit the value of HPC_{max}. Note also that as we scale to smaller feature sizes, cores shrink while die sizes remain unchanged, so the same interconnect length will translate to a larger HPC_{max}.

Full-system traffic

Full-system simulations use Wind River Simics within GEMS,[12] with 64 in-order Sparc cores. We model 32-Kbyte private instruction and data L1 caches, and a 1-Mbyte L2 cache slice per tile. We evaluate the parallel sections of Splash-2[14] and Parsec[15] for both private and shared L2, over a MOESI directory protocol. Each run consists of 64 threads of the application running on our chip multiprocessor.

Figure 10 shows that Smart-8_1D and Smart-8_2D lower application runtime by 26 and 27 percent, respectively, on average, for a private L2 where only L2 misses traverse the network; this is only 8 percent away from an ideal ($T_N = 1$) network. The runtime reduction goes up to 49 and 52 percent, respectively, with a shared L2 design, where both L1 and L2 misses traverse the network (making network latency even more critical), which is 9 percent off from an ideal ($T_N = 1$) network. Smart-15_2D does not give any significant runtime benefit over Smart-8_2D.

Aggressive NoC pipeline optimizations can lower router delays to just one cycle. However, this is not good enough for large networks with multihop 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 multihop paths within a single cycle by virtually bypassing all routers along the route, without adding any physical channels on the datapath. In the best case, the network latency with Smart, from Equation 2, is just 2 cycles if there is no contention (that is, for all h, $t_c(h) = 0$), and H is less than HPC_{max}. In the worst case of $t_c(h) > 0$ at every hop h, the achieved HPC will be 1, which is the same as the baseline.

Although transistors become faster with technology scaling, wires do not. This trend of communication becoming slower relative to logic has been projected in the past as a motivation to keep global chip-wide communication to a minimum and heavily optimize for locality. This work rebuts this conclusion. We project that communication (wire) delay in cycles will actually remain relatively constant as technology scales, as chip dimensions are not increasing due to yield, and clock frequencies have also plateaued owing to the power wall. In addition, tile sizes tend to go down as technology scales. The same wire delay will thus translate to a higher HPC_{max} as technology scales, making Smart even more attractive. Locality will no longer be that critical with Smart NoCs.

This work opens up a plethora of research opportunities in circuits, NoC architectures, and locality-oblivious many-core architectures to optimize and leverage Smart NoCs. MICRO

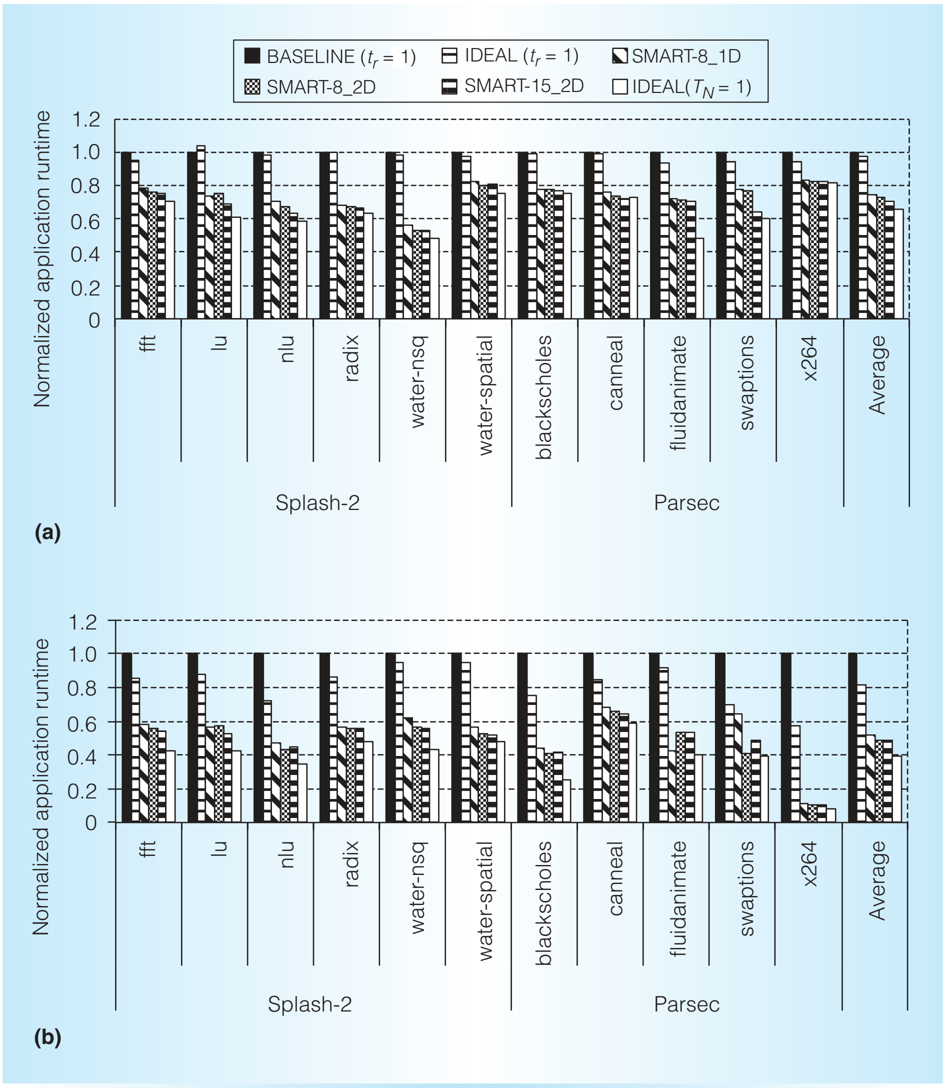

Figure 10. Full-system application runtime with Smart, normalized to the runtime with baseline($t_r = 1$). Private L2 cache per tile (a). Shared L2 cache slice per tile (b). In Shared L2, L1 and L2 misses traverse the network to a remote node, making on-chip network latency more critical than in Private L2.

Acknowledgments

We thank Sunghyun Park from the Massachusetts Institute of Technology and Michael Pellauer from Intel for useful insights on the interconnect and pipeline. We acknowledge the support of DARPA UHPC, SMART LEES, and MARCO C-FAR.

References

1. W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2003.
2. A. Kumar et al., "A 4.6Tbits/s 3.6GHz Single-Cycle NoC Router with a Novel Switch Allocator in 65nm CMOS," *Proc. 25th Int'l Conf. Computer Design*, 2007, pp. 63-70.
3. R. Mullins et al., "Low-Latency Virtual-Channel Routers for On-Chip Networks," *Proc. 31st Ann. Int'l Symp. Computer Architecture*, 2004, pp. 188-197.
4. H. Matsutani et al., "Prediction Router: Yet Another Low Latency On-Chip Router Architecture," *Proc. IEEE 15th Int'l Symp. High Performance Computer Architecture* (HPCA 09), 2009, pp. 367-378.

5. A. Kumar et al., "Token Flow Control," *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture*, 2008, pp. 342-353.
6. S. Park et al., "Approaching the Theoretical Limits of a Mesh NoC with a 16-Node Chip Prototype in 45nm SOI," *Proc. 49th Ann. Design Automation Conf.* (DAC 12), 2012, pp. 398-405.
7. Y. Hoskote et al., "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, 2007, pp. 51-61.
8. J. Howard et al., "A 48-core IA-32 Message-Passing Processor with DVFS in 45nm CMOS," *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2010, pp. 108-109.
9. B. Kim and V. Stojanovic, "Equalized Interconnects for On-Chip Networks: Modeling and Optimization Framework," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design* (ICCAD 07), 2007, pp. 552-559.
10. J. Rabaey and A. Chandrakasan, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, 2002.
11. C. Sun et al., "DSENT—A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," *Proc. IEEE/ACM 6th Int'l Symp. Networks-on-Chip*, 2012, pp. 201-210.
12. M.M.K. Martin et al., "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, 2005, pp. 92-99.
13. N. Agarwal et al., "GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software*, 2009, pp. 33-42.
14. S.C. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, 1995, pp. 24-36.
15. C. Bienia et al., "The PARSEC Benchmark Suite: Characterization and Architectural Implications," *Proc. 17th Int'l Conf. Parallel Architectures and Compilation Techniques* (PACT 08), 2008, pp. 72-81.

Tushar Krishna is a researcher in the VSSAD group at Intel. His research focuses on on-chip interconnection networks for homogeneous and heterogeneous many-core systems. Krishna has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology, where he performed the work for this article.

Chia-Hsin Owen Chen is a doctoral candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His research interests include system-level power and performance modeling and analysis, and on-chip networks. Chen has an SM in electrical engineering and computer science from the Massachusetts Institute of Technology.

Woo-Cheol Kwon is a doctoral candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His research interests include multicore processor architectures and networks on chip. Kwon has an MS in computer science from the Korea Advanced Institute of Science & Technology (KAIST).

Li-Shiuan Peh is a professor in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. Her research focuses on networked computing in many-core chips and mobile wireless systems. Peh has a PhD in computer science from Stanford University. She is a member of IEEE and the ACM.

Direct questions and comments about this article to Tushar Krishna, 77 Reed Road, HD2-330, Hudson, MA 01749; tushar.krishna@intel.com.