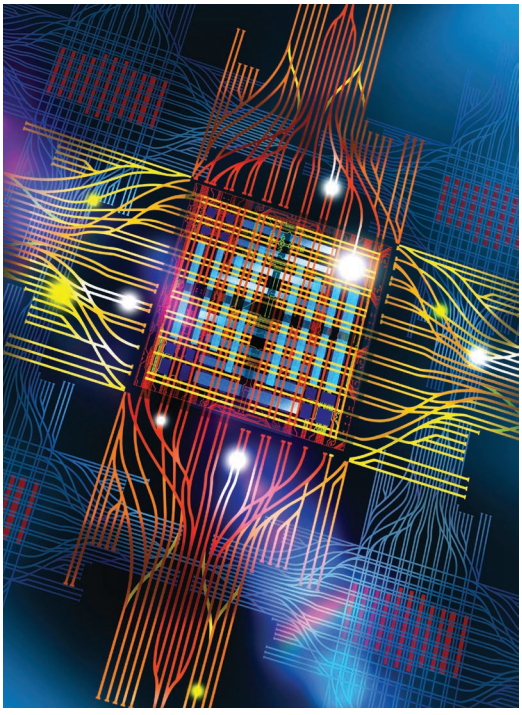

Single-Cycle Multihop Asynchronous Repeated Traversal: A SMART Future for Reconfigurable On-Chip Networks

Tushar Krishna, Chia-Hsin Owen Chen, Sunghyun Park, Woo-Cheol Kwon, Suvinay Subramanian, Anantha P. Chandrakasan, and Li-Shiuan Peh, *MIT*

Future scalability for kilo-core architectures requires solutions beyond the capabilities of protocol and software design. Single-cycle multihop asynchronous repeated traversal (SMART) creates virtual single-cycle paths across the shared network between cores, potentially offering significant reductions in runtime latency and energy expenditure.

Today's multicore chip architectures require scalable network topologies, such as meshes, that facilitate communication between cores. A scalable on-chip network connects cores by means of a wire grid, or mesh, with crosspoint routers that orchestrate this communication flow, arbitrating the sharing of link segments and buffering messages upon contention. The smallest unit of a network message, or flit, traverses a series of router and link segments, or hops, from its source to its destination. The length of each of a flit's hops is the tile width, the distance between two routers.

The equation for the network latency (T) of a flit is[1]

$$T = H(t_r + t_w) + T_c,$$

where H is the number of hops; t_r is the router pipeline delay; t_w is the wire (the link between two routers) delay, which is typically a single cycle; and T_c is the network contention delay. Microarchitecture research over the last decade has lowered t_r from five cycles to a single cycle[2] so that the low-load network latency between a source core and a destination core is equal to the number of routers plus the number of link segments (that is, the number of hops times 2) between the two cores.

As the number of cores increases, however, H inevitably increases, increasing T proportionately. Higher on-chip latency not only delays initial request and response time, but because of dependencies also bottlenecks additional requests and responses, which leads to poorer throughput and overall system slowdown. For this reason—that is, in order to minimize average network hops—coherence protocol designers prefer private L2 architectures, and programmers and compiler/operating system designers try to map data close to sharers. But protocol and software design can go only so far: ultimately, each core has a limited number of single-hop neighbors, and the more cores, the more hops required for message traversal across

 Published by the IEEE Computer Society

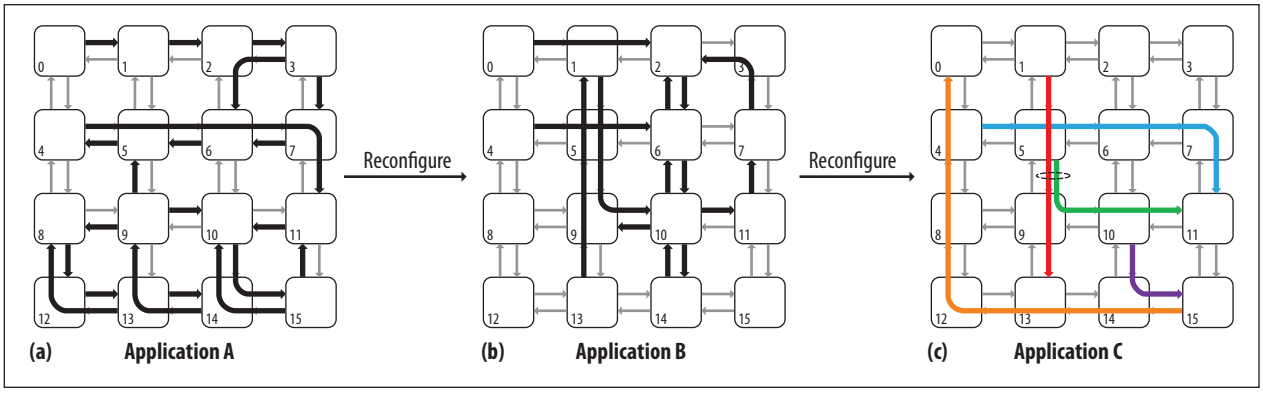

Figure 1. Example of single-cycle multihop asynchronous repeated traversal (SMART) across three traffic scenarios, A, B, and C. All the multihop paths in boldface are traversed in one cycle. For application C, the flows are colorized to show that the red and green flows contend for the shared link between routers 5 and 9.

the chip network. Based on current limitations, designing for an exascale goal of 1,024 cores per chip will result in horrendous levels of on-chip network latency and energy expenditure, slowing the whole system.

To mitigate this problem, we propose single-cycle multihop asynchronous repeated traversal (SMART), a paradigm that removes the dependence of T on H by creating virtual single-cycle multihop routes—paths constituting a single cycle across the network, from source to destination, irrespective of the physical number of hops between cores. The fundamental methodology is to drive signals through multiple routers and links asynchronously, without latching. Our evaluations show that SMART can reduce application runtime by 61 percent for system-on-chip (SoC) traffic and 50 to 57 percent across private and shared L2 cache coherence traffic.

FUNDAMENTALS OF SMART

Conventionally, on-chip latency depends on H because messages are latched at every router. Router logic delay limits network frequency to between 1 and 2 GHz at 45 nm.[2,3] Link drivers are sized accordingly, to drive signals one hop (assumed here to be 1 mm) in 0.5 to 1 ns before the signal is latched at the next router.

However, as the sidebar "Clockless Repeated Signaling" demonstrates, wires themselves can potentially traverse multiple millimeters within the same frequency (1-2 GHz). SMART decouples wires from routers by replacing the clocked link drivers at each router with clockless repeaters. As our previous research shows, this allows flits to traverse multiple hops—from 1 hop to HPC_{max} (maximum hops per cycle) hops—within a single cycle before getting latched, where HPC_{max} is 16 for a network data path with 1-mm tiles at 1 GHz.[4] Effectively, we reduce the value of H in the equation $T = H(t_r + t_w) + T_c$ to $\lceil (H/HPC_{max}) \rceil$.

The network we propose can reconfigure and create virtual single-cycle paths across multiple hops for any traffic pattern as shown in Figure 1, where three sets of traffic flows incur single-cycle paths, as long as no contention occurs for the same links. This reconfiguration involves presetting the routers along each route at runtime. We analyzed reconfigurations at two time granularities: $SMART_{app}$, which reconfigures SMART on an application-by-application time granularity, keeping settings static during each application's runtime; and $SMART_{cycle}$, which reconfigures SMART on a cycle-by-cycle time granularity, altering settings during each cycle to adapt to actual traffic.

SMART ROUTERS

Figure 2a shows the microarchitecture of a SMART router. For simplicity, we show only the core-in (C_{in}), west-in (W_{in}), and east-out (E_{out}) ports. All other input ports (north-in and south-in) are identical to W_{in}, and all other output ports are identical to E_{out}.

The three primary components of the design are the *buffer write enable* (BW_{ena}) at the input flip-flop, which determines whether the input signal is latched or not; the *bypass multiplexer select* (BM_{sel}) at the input of the crossbar switch, which chooses between the local (buffered) flit and the bypassing flit on the input link; and the *crossbar select* (XB_{sel}), which connects an input port to an output port. These are the SMART control signals. If BW_{ena} is set to *0* and BM_{sel} is set to *bypass*, the bypass path is enabled; therefore, the incoming flit moves directly to the crossbar and through the repeater to the output link, without being buffered (that is, latched) at the router. If BW_{ena} is set to *1*, on the other hand, the input flit is latched and buffered; it now needs to compete with other buffered flits within the router to gain access to the crossbar switch, called *switch arbitration local* (SA-L).

Figure 2b shows an example of a multihop traversal: a flit from router R0 traverses three hops within a single cycle, eventually latching at router R3. In R0, BM_{sel} is preset to *local*. The crossbars at R1 and R2 are preset to connect W_{in} to E_{out}, with their BM_{sel} preset to *bypass*, and BW_{ena} preset to *0*. In R3, BW_{ena} is preset to *1*. Each single-cycle

Clockless Repeated Signaling

A standard means of reducing delay of long wires is to insert repeaters (that is, a pair of inverters) in the wire at regular intervals.[1] We performed an experiment to measure the maximum length a signal can traverse on such a repeated wire. We laid out parallel repeated wires in 45 nm, with a repeater spacing of 1 mm (our assumed tile size) and wire spacing of three times the minimum possible wire spacing allowed by the technology (to reduce coupling capacitance and thereby increase speed). We then increased the length of the wire incrementally, using a commercial place-and-route tool to size the repeaters so as to achieve 1-ns timing at each data point. Figure A plots the energy consumed at each data point. We can see that a signal on the repeated wire can traverse a length of up to 16 mm within a nanosecond at 45 nm. We translate this distance to a network microarchitectural parameter maximum hops per cycle (HPC_{max}), defined as

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

At 45 nm, assuming a tile size of 1 mm × 1 mm and a target clock period of 1 ns (or 1 GHz), 16 mm/ns (Figure A) translates to an HPC_{max} of 16. We replace conventional clocked drivers with repeaters at every router, thus allowing flits to traverse up to HPC_{max} hops within a 1-GHz clock. Moreover, we can see that clockless repeated wires consume 14.3 percent lower energy/bit/mm than clocked repeated wires consume.

As technology scales, repeated wire delay is expected to remain relatively constant.[2,3] It makes sense intuitively that as feature size decreases, wires will become thinner, causing resistance to go up and capacitance-to-ground to go down, thus keeping wire delay approximately the same. Moreover, chip dimensions should remain essentially the same (~20 mm × 20 mm) due to yield, and clock frequencies will almost certainly remain constant to stay within the chip's fixed power budget. These two projections, coupled with expected projections of relatively constant wire delay and the smaller tile size resulting from scaling, suggest that HPC_{max} will likely increase with future technologies.

References

1. J. Rabaey and A.P. Chandrakasan, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, 2002.
2. T. Krishna et al., "Breaking the On-chip Latency Barrier Using SMART," *Proc. 19th Int'l Symp. High-Performance Computer Architecture* (HPCA 13), IEEE, 2013, pp. 378-389.
3. R. Ho, K.W. Mai, and M.A. Horowitz, "The Future of Wires," *Proc. IEEE*, vol. 89, no. 4, 2001, pp. 490-504.

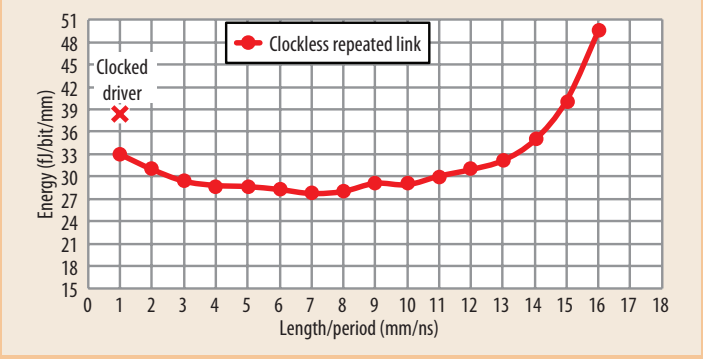

Figure A. Energy consumed by clockless repeated links as a function of distance traversed (mm) per nanosecond at 45 nm.

multihop path, or SMART path, can be created by appropriately setting BW_{ena}, BM_{sel}, and XB_{sel} at the intermediate routers. The SMART control signals are preset before the actual message arrives, and this presetting can be accomplished at different time granularities, either prior to application runtime ($SMART_{app}$) or cycle-by-cycle ($SMART_{cycle}$), as mentioned earlier.

$SMART_{app}$: application-by-application reconfiguration

Traffic flows in special-purpose multiprocessor on-chip systems (MPSoCs), such as the chips that run smartphones, and those in heterogeneous multicores with general purpose accelerators are well established. Because we know these flows in advance, we can reconfigure the network before running the application, creating single-cycle SMART paths between nodes that communicate regularly and are physically far apart, based on an application's task communication graph. We discuss the details of this reconfiguration stage elsewhere.[5]

In these cases, BW_{ena}, BM_{sel}, and XB_{sel} are set according to the existence of any overlapping, or conflicting, flows through an output link. For example, in Figure 1c the orange, blue, and purple flows do not conflict at any link, so the routers on these flows are preset to *bypass* (BW_{ena} = *0* and BM_{sel} = *bypass*), and XB_{sel} is appropriately preset, thus creating a single-cycle path all the way from the source to the destination throughout the runtime of the application. The red and green flows, on the other hand, overlap at the link between routers 5 and 9, and so need to stop at both these routers (that is, BW_{ena} = 1), arbitrate for the crossbar's south output port at router 5 and north input port at router 9, and set XB_{sel} accordingly. The remaining segments of their traversal consist of a single cycle.

$SMART_{cycle}$: cycle-by-cycle reconfiguration

While appropriate for MPSoCs, $SMART_{app}$ will not work in general-purpose multicore chips because precise prediction of each application's traffic flow is not possible; any core could potentially communicate with any other core via cache coherence memory traffic. $SMART_{cycle}$ addresses such contingencies by allowing the creation of different SMART paths on a cycle-by-cycle basis. Here, the method is first to set up a multihop path and then traverse it; setting

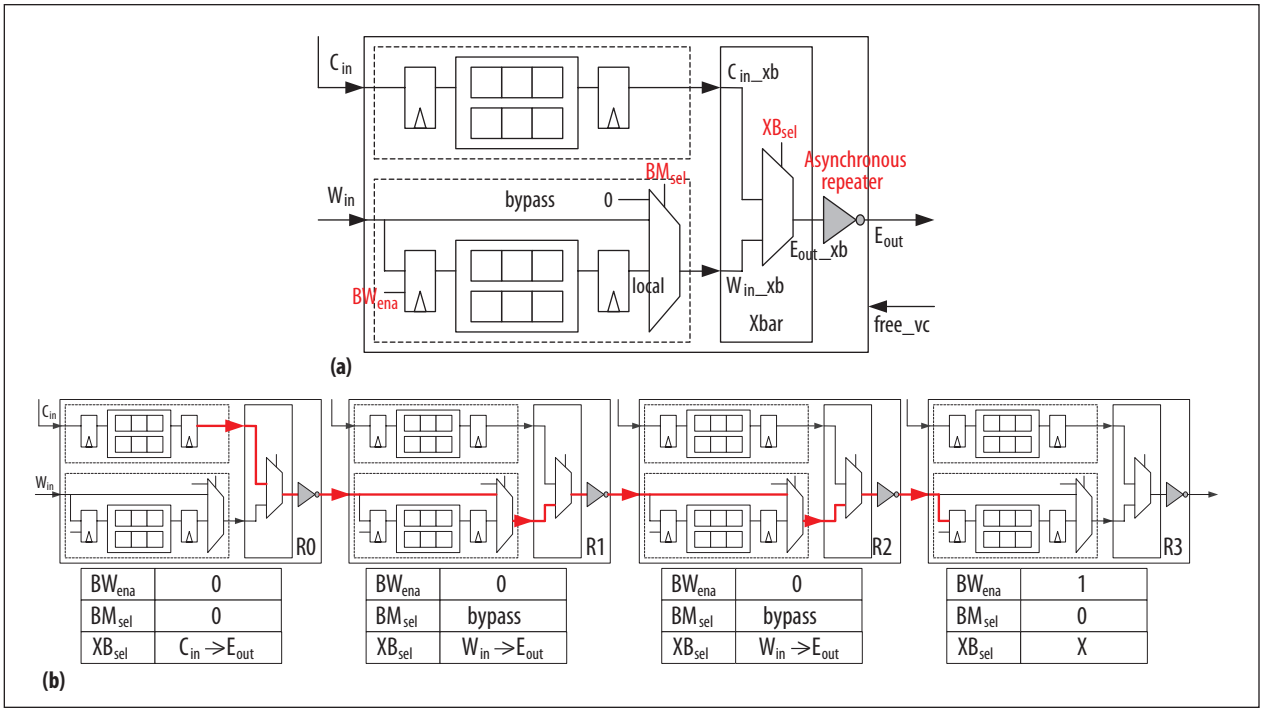

Figure 2. SMART router microarchitecture, with an example of single-cycle, multihop traversal. (a) The three primary components of the SMART router design are the buffer write enable (BW_{ena}), the bypass multiplexer select (BM_{sel}), and the crossbar select (XB_{sel}). (b) A flit from router R0 traverses R1 and R2 before latching at R3, according to each router's BW_{ena}, BM_{sel}, and XB_{sel} settings. C_{in}: core-in port; W_{in}: west-in port; E_{out}: east-out port; vc: virtual channel.

up the path takes one cycle, and traversing it takes another cycle.

$SMART_{cycle}$ network traversal. The first cycle comprises SA-L arbitration, as each router chooses a winner from among its buffered local flits for each output port. The next cycle sets up the multihop route: the winner at each output port broadcasts a *SMART-hop setup request* (SSR) of up to HPC_{max} hops from that output port. These SSRs—dedicated repeated wires that connect every router to a neighborhood of up to the HPC_{max}—help preset the SMART control signals at all intermediate routers. SSRs are $\log_2(1 + HPC_{max})$ bits wide, and carry the number of hops the flit wishes to traverse. For example, $SSR = 2$ indicates a 2-hop path request. Each flit sends out an SSR that is as close as possible to its final destination.

Following the initial SSR broadcast, every router performs a second round of arbitration, *switch allocation global* (SA-G), to arbitrate among the SSRs they have received from the routers in their HPC_{max} neighborhood and to set the SMART control signals accordingly. The SA-G arbiters guarantee that only one flit will be allowed access to any particular input/output port of a router crossbar; any conflicting flits will be stopped. Every router prioritizes SSR requests according to a fixed priority based on flit distance, where the local flit has highest priority, followed by the flit from the closest neighboring router, followed by the flit from the router two hops away, and so on.

In the ensuing cycle, SA-L winners traverse crossbars and links for multiple hops until they are stopped at a router where $BW_{ena} = 1$. Thus, flits spend two cycles (SA-L arbitration, followed by SSR + SA-G arbitration) at a router before they traverse a single-cycle multihop path.

Figure 3 provides an illustration. Here, 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 arbitration during Cycle 0. In Cycle 1, it sends out $SSR_D = 2$ (that is, a request to stop at R4) out of E_{out} to routers R3, R4, and R5. Each of these routers then performs SA-G arbitration. At R2, which is 0 hops away ($< SSR_D$), BM_{sel} = local, and $XB_{sel} = W_{in} \rightarrow E_{out}$. At R3, which is 1 hop away ($< SSR_D$), BM_{sel} = bypass, and $XB_{sel} = W_{in} \rightarrow E_{out}$. At R4, which is the requested 2 hops away ($= SSR_D$), $BW_{ena} = 1$. 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.

Competing SSRs. Using the same example, suppose router R0 simultaneously wants to send $Flit_E$ three hops away to R3 ($SSR_E = 3$), as shown in Figure 4. In Cycle 1, router R2 sends out SSR_D as before, and in addition R0 sends SSR_E out of E_{out} to R1, R2, and R3. At R2 a conflict now occurs between SSR_D and SSR_E for the W_{in} and E_{out} ports of the crossbar. Using our priority scheme, $Flit_E$ loses to $Flit_D$ because local flits have the highest priority within a router. In Cycle 2, $Flit_E$ traverses the crossbar and link

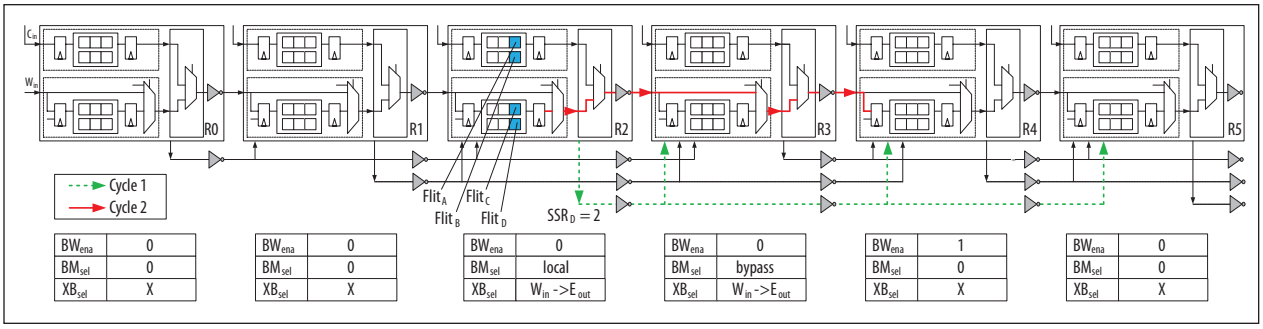

Figure 3. SMART$_{cycle}$ example with no SMART-hop setup request (SSR) conflicts among flits for the same path.

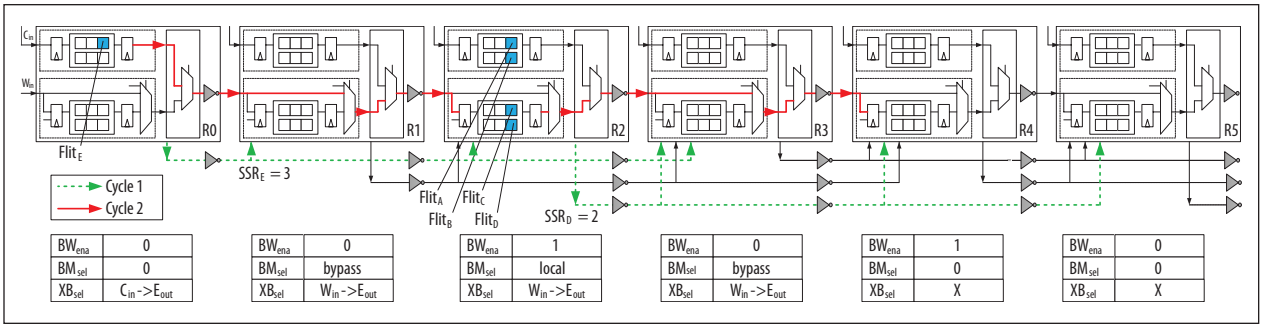

Figure 4. SMART$_{cycle}$ example with SSR conflicts among flits.

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 SA-L arbitration at R2 before it can send a new SSR and continue its network traversal.

It is clear, then, that all paths in SMART$_{cycle}$ are opportunistic. Barring any contention for a set of links during a cycle, a flit can create and traverse single-cycle paths all the way to its destination, regardless of the physical number of hops (as long as this number is within HPC_{max}), just as with SMART$_{app}$ paths. But if there is contention for a link or for multiple links, flits must stop and serialize over them. In other words, in the worst-case patterns, SMART$_{cycle}$ behaves in the same manner as a conventional baseline network. The same is true if traffic is always sent to the nearest one-hop neighbor and multihop paths are not required.

Implementation and overheads. In contrast to SMART$_{app}$, where an offline computation offers more flexibility for creating contention-free routes between routers, in SMART$_{cycle}$, the physical layout of the SSR wires from each router determines route flexibility and HPC_{max}. We saw in the "Clockless Repeated Signaling" sidebar that repeated wires have an HPC_{max} of 16. However, the critical path of the design is $\min\{T_{SSR+SA\text{-}G}, T_{ST+LT}\}$ where SSR + SA-G refers to the path setup stage, while ST + LT refers to the switch + link traversal stage. SSR and LT are pure repeated wire traversals, whereas SA-G and ST involve logic computation. We observe that ST + LT lowers HPC_{max} to 11. The delay of SA-G depends on the number of SSRs entering the router. SSRs spanning a single dimension (that is, X+, X−, Y+, and Y−) restrict flits from bypassing turning routers and lower HPC_{max} to 13. SSRs that span both dimensions allow flits to bypass at turns, but at the cost of a more complex SA-G lowering HPC_{max} to 9.

The SSR bits and clockless repeaters add negligible area overhead because router area is determined by the wire-dominated crossbar.[2]

EVALUATION

Here, we evaluate SMART using two traffic scenarios: SoC application traffic and chip-level multiprocessing (CMP) cache coherence traffic. (Elsewhere, we have described evaluations across additional traffic scenarios.[4]) In this discussion, SMART is compared against two yardsticks:

- Our *baseline* is a router where $t_r = 1$; in other words, each flit spends one cycle in the router (although more in case of contention), followed by one cycle in the link, yielding two cycles per hop. This is currently the best available network design.
- Our *ideal* is an imagined network where every flit travels from the source router to the destination router (that is, across the network) within one cycle, with no contention at all along the route. This is the best that SMART can potentially accomplish.

All designs are modeled in the Wisconsin Multifacet General Execution-driven Multiprocessor Simulator (GEMS)[6] within the Garnet network model.[7] We target a 45-nm technology node and 1-GHz clock frequency. (For purposes of comparison, the sidebar "Related Work" surveys other proposed methodologies with goals similar to those of SMART.)

SoC application traffic

To test the impact of $SMART_{app}$, we ran traces from eight SoC applications individually through a 4 × 4 mesh network. Our reconfiguration algorithm uses the task communication graph for each application as input, and creates a virtual topology overlaid over the mesh, optimizing for as many contention-free routes as possible[4] to minimize stopping. For example, Figure 1a shows the topology mapping for the *vopd* benchmark. The graph in Figure 5 compares the performance of $SMART_{app}$ with our established baseline and ideal systems. Compared to the baseline, $SMART_{app}$ reduced network latency by 60.1 percent on average. The average network latency using $SMART_{app}$ was 3.8 cycles, only 1.5 cycles higher than that of the ideal single-cycle topology; for *pip*, *vopd*, and *wlan*, the latencies achieved by $SMART_{app}$ were almost identical to those of the ideal (latency here is more than one cycle because of serialization among flows at the link from the destination router to the core).

In *h264* and *mms_mp3*, however, where one core acts as a sink for most flows while another acts as the source for most flows, heavy contention results in multiplexing of different flows on the shared links in $SMART_{app}$; our ideal, on the other hand, has no bandwidth limitation, and so performs between two and four cycles faster.

In our test runs, we changed the task-to-core mapping for every application. In an actual SoC, this mapping may not be able to change drastically across applications, as cores are often heterogeneous and certain tasks are tied to specific cores. This results in longer paths, which would magnify the benefits of $SMART_{app}$.

Full-system cache coherence traffic

To study the impact of SMART in a many-core cache coherence traffic environment, we ran full-system 64-threaded SPLASH-2 and PARSEC applications on an 8 × 8 mesh. We used 32-kB instruction and data (I & D) private L1 caches per core, and distributed an aggregate L2 capacity of 16 MB (comparable to the last-level cache capacities of commercial Intel/AMD chips currently available) across the 64 tiles.

Because any core can communicate with any other core, $SMART_{app}$ is not useful in these environments; it cannot find any contention-free routes during the reconfiguration phase, forcing flits to stop and arbitrate at every hop, which leads to performance identical to that of the baseline. Thus we evaluate $SMART_{cycle}$ in these environments.

We compare both a private and a shared L2 design, running a MOESI directory protocol. In private L2 designs, a copy of the data is retained in the local L2 within the tile for fast access upon future L1 misses. However, replication of

Related Work

Physical express topologies such as clos[1] and flattened butterfly,[2] among others, advocate adding physical express links between distant routers to reduce average hops. These express point-to-point links can be further engineered for lower delay using repeaters. Each router now has >5 ports, and channel bandwidth is often reduced proportionally to offer buffer and crossbar area/power similar to a mesh (radix-5) router.

More ports, however, imply a complicated routing and arbiter mechanism, with a hierarchical arbiter and crossbar,[3] increasing router delay t_r to between 4 and 5 cycles at the routers where flits do need to stop. 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 traffic) because of higher serialization delay.

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.

Virtual express topologies are similar to SMART in that they preset a router's input port and crossbar such that an incoming flit can bypass buffering and proceed straight to the crossbar and on to the link. However, most prior proposals—both in the SoC domain, such as VIP[4] and ReNoC,[5] and in the CMP domain, such as EVC[6] and TFC[7]—have focused on realizing a single cycle per hop, rather than a single cycle across multiple hops. SMART, on the other hand, is designed for single-cycle network traversals, and is the first work to perform reconfiguration across multiple hops within the same cycle.

References

1. Y.H. Kao et al., "CNoC: High-Radix Clos Network-on-Chip," *IEEE Trans. Computer-Aided Design Integrated Circuits and Systems* (TCAD 11), vol. 30, no. 12, 2011, pp. 1897-1910.
2. J. Kim, J. Balfour, and W.J. Dally, "Flattened Butterfly Topology for On-Chip Networks," *Proc. 40th IEEE/ACM Int'l Symp. Microarchitecture* (MICRO 07), IEEE, 2007, pp. 172-182.
3. J. Kim et al., "Microarchitecture of a High-Radix Router," *Proc. 32nd IEEE/ACM Int'l Symp. Computer Architecture* (ISCA 05), IEEE, 2005, pp. 420-431.
4. M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Virtual Point-to-Point Connections for NoCs," *IEEE Trans. Computer-Aided Design Integrated Circuits and Systems* (TCAD), vol. 29, no. 6, 2010, pp. 855-868.
5. M.B. Stensgaard and J. Sparsø, "ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology," *Proc. 2nd ACM/IEEE Int'l Symp. Networks-on-Chip* (NOCS 08), IEEE, 2008, pp. 55-64.
6. A. Kumar et al., "Express Virtual Channels: Towards the Ideal Interconnection Fabric," *Proc. 34th IEEE/ACM Int'l Symp. Computer Architecture* (ISCA 07), IEEE, 2007, pp. 150-161.
7. A. Kumar, L.-S. Peh, and N.K. Jha, "Token Flow Control," *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture* (MICRO 08), IEEE, 2008, pp. 342-353.

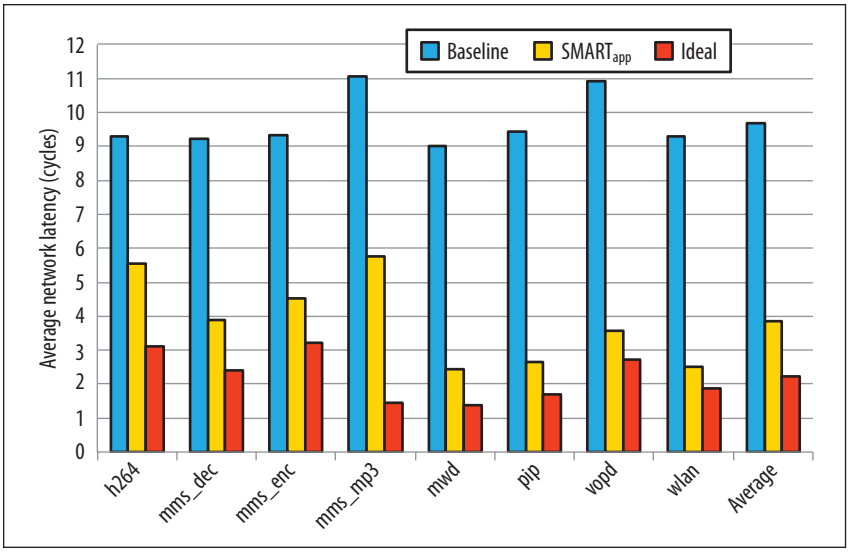

Figure 5. Performance evaluations comparing SMART$_{app}$ with a baseline (two-cycle per hop) network and an ideal (one-cycle across the network) network for a suite of eight SoC applications.

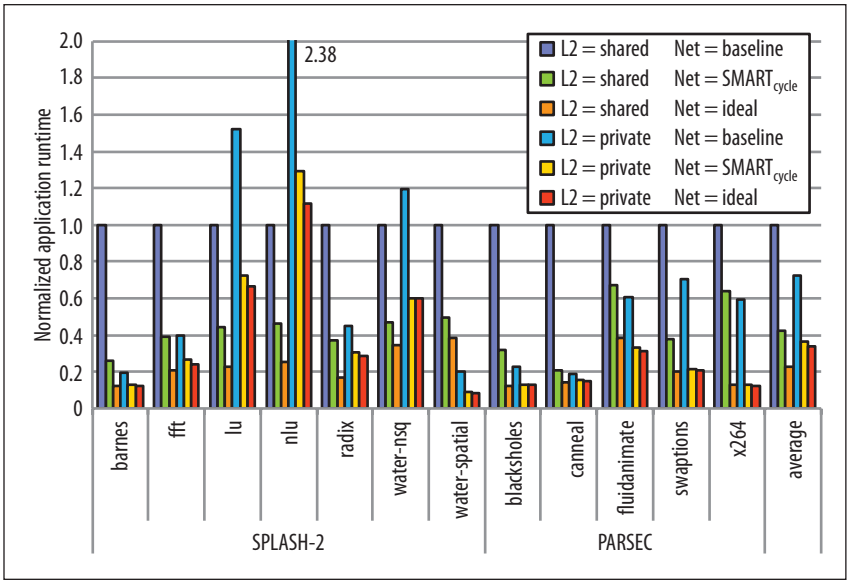

Figure 6. Full-system runtime comparisons of SMART$_{cycle}$ with baseline and ideal networks in shared L2 and private L2 environments. Seven applications from SPLASH-2 and five applications from PARSEC are shown.

data across tiles lowers the effective on-chip cache capacity. In shared L2 designs, there is only one copy of the data on-chip, increasing cache capacity. However, L1 misses always involve a network traversal to access data at a remote L2, making on-chip latency critical to performance.

Figure 6 plots the application runtime (normalized to the shared L2 with a baseline network). In our imagined ideal network (with constant one-cycle network delay), a private L2 is 50.4 percent slower than a shared L2 on average as a result of reduced cache capacity. However, in an actual baseline network, remote accesses become costly, making a private L2 27.8 percent faster on average than a shared L2 and thus more favorable, despite the reduction in overall cache capacity.

SMART completely reverses these tradeoffs. SMART$_{cycle}$ lowers application runtime by 57.5 percent in a shared L2 and by 49.5 percent in a private L2 design compared with the respective baseline networks. For almost all benchmarks, a shared L2 design with a SMART$_{cycle}$ network matches or even betters performance (*lu*, *nlu*, *radix*, *water-nsq*, *swaptions*) when compared to the private L2 design running on the baseline network, because the necessarily frequent remote accesses of shared caches are no longer as expensive. These results suggest that SMART can potentially enable a hybrid private/shared L2 design that can both avoid unnecessary replication and lower the remote access time, thereby providing scalability.

As core counts increase, on-chip network latency inevitably plays a crucially important role in determining the performance of shared memory systems. Traditionally, on-chip latency has been proportionate to the number of hops in a traversal. Research in efficient network design has helped lower per-hop latency to between one and two cycles, but this will not suffice as we scale toward chips of 1,000 cores and higher. SMART addresses on-chip latency by proposing a technique for traversing multiple hops within a single cycle, driving links by asynchronous repeaters at every hop and reconfiguring the switches connecting these links, either before the application is run or while the application is running, depending on the target traffic domain.

Today, repeated wires can transmit signals up to 16 mm within 1 ns. With maximum chip sizes of ~20 mm × 20 mm due to yield and cost, this means that SMART can enable single-cycle traversals across the chip. As technology scales, tile sizes will go down, while global wire delay will likely remain the same, meaning that we can potentially cross even more hops within a single cycle.

A SMART network not only reduces on-chip latency but also reopens the debate concerning private versus shared L2 cache capacity. Thus, SMART can potentially pave the way for locality-oblivious multicore architectures, ease the burden on OS/compiler and protocol designers, and help usher in the era of kilo-core chips.

References

1. W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2003.
2. S. Park et al., "Approaching the Theoretical Limits of a Mesh NoC with a 16-Node Chip Prototype in 45-nm SOI," *Proc. 49th Design Automation Conf.* (DAC 12), IEEE, 2012, pp. 398-405.
3. J. Howard et al., "A 48-Core IA-32 Message-passing Processor with DVFS in 45-nm CMOS," *Proc. Int'l Solid State Circuits Conf.* (ISSCC 10), IEEE, 2010, pp. 108-109.
4. T. Krishna et al., "Breaking the On-chip Latency Barrier Using SMART," *Proc. 19th Int'l Symp. High-Performance Computer Architecture* (HPCA 13), IEEE, 2013, pp. 378-389.
5. C.H.O. Chen et al., "SMART: A Single-Cycle Reconfigurable NoC for SoC Applications," *Proc. Design Automation and Test in Europe* (DATE 13), 2013, pp. 338-343.
6. 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.
7. 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* (ISPASS 09), IEEE, 2009, pp. 33-42.

Tushar Krishna *is a doctoral candidate in the Department of Electrical Engineering and Computer Science at MIT. His research interests include on-chip interconnection networks for parallel many-core systems. He received a BTech from the Indian Institute of Technology, Delhi, and an MSE from Princeton University. Krishna is a student member of IEEE. Contact him at tushar@csail.mit.edu.*

Chia-Hsin Owen Chen *is a doctoral candidate in the Department of Electrical Engineering and Computer Science at MIT. His research interests include system-level power and performance modeling and analysis in on-chip networks. He received a BS from National Taiwan University and an SM from MIT. Contact him at owenhsin@csail.mit.edu.*

Sunghyun Park *is a doctoral candidate in the Department of Electrical Engineering and Computer Science at MIT. His research interests include energy-scalable network-on-chip design at circuit, VLSI-CAD, and architecture levels. He received a BS from the Korea Advanced Institute of Science and Technology and an SM from MIT. Park is a recipient of the Samsung Scholarship. Contact him at pshking@mit.edu.*

Woo-Cheol Kwon *is a doctoral candidate in the Department of Electrical Engineering and Computer Science at MIT. His research interests include multicore processor architecture and network on chip. He received a BS and MS from the Korea Advanced Institute of Science and Technology. Before joining MIT, he worked for Samsung Electronics; where he conducted system interconnect design, performance analysis, and functional verification. Contact him at wckwon@mit.edu.*

Suvinay Subramanian *is a doctoral candidate in the Department of Electrical Engineering and Computer Science at MIT. His research interests include on-chip interconnection networks and computer architecture. He received a BTech from the Indian Institute of Technology, Madras, and an SM from MIT. Contact him at suvinay@csail.mit.edu.*

Anantha P. Chandrakasan *is the Joseph F. and Nancy P. Keithley Professor of Electrical Engineering at MIT. His research interests include micropower digital and mixed-signal integrated circuit design, wireless microsensor system design, portable multimedia devices, energy-efficient radios, and emerging technologies. Chandrakasan holds a PhD from the University of California, Berkeley. His numerous awards include the 2009 Semiconductor Industry Association (SIA) University Researcher Award and the 2013 IEEE Donald O. Pederson Award in Solid-State Circuits. Contact him at anantha@mtl.mit.edu.*

Li-Shiuan Peh *is a professor in the Department of Electrical Engineering and Computer Science at MIT. Her research focuses on networked computing in many-core chips as well as mobile wireless systems. Peh received a PhD in computer science from Stanford University. She was awarded the ACM Distinguished Scientist Award in 2011, the CRA Anita Borg Early Career Award in 2007, and the Sloan Research Fellowship in 2006. She is a member of IEEE and ACM. Contact her at peh@csail.mit.edu.*