Reactive Circuits: Dynamic Construction of Circuits for Reactive Traffic in Homogeneous CMPs

Marta Ortín-Obón[§], Darío Suárez-Gracia[§], María Villarroya-Gaudó[§], Cruz Izu[∓], and Víctor Viñals[§]

[§] *Departamento de Informática e Ingeniería de Sistemas, University of Zaragoza, Spain*
[∓] *School of Computer Science, University of Adelaide, Australia*

Abstract

Networks on Chip (NoCs) have a large impact on system performance, area, and energy. NoCs convey request and response messages among cores following the message patterns dictated by the cache banks. Such patterns do not only guarantee a coherent memory state, but also provide an opportunity for NoC optimization. Request messages can smartly reserve the resources to dynamically build a circuit for replies, thus reducing their network latency. Starting from this simple idea, which we denote Reactive Circuits, we evaluate several implementations of the mechanism: with and without sharing circuits between messages, performing timed reservations, and removing the implicit coherence messages. A careful implementation of this circuit reservation mechanism in a wormhole router achieves an average 20.8% reduction in network energy consumption, 5.8% smaller router area and a 4.8% system performance increase in a 64-core chip, compared with a conventional network.

Keywords: Chip Multiprocessor, Interconnection Network, Coherence Protocol

1. Introduction

Chip multiprocessors (CMPs) are now a common design to improve performance by exploiting thread parallelism. They are built by replicating simple small cores that share a coherent memory space and are connected via an interconnection network. All the components of these chips must be carefully co-designed to achieve the desired performance while maintaining

reasonable power consumption. We focus on optimizing the network on-chip (NoC) by customizing it for the traffic it has to support, keeping in mind that the most relevant metric we must take care of is latency [1]. The traffic is generated by the coherence protocol to transport information among caches located in different nodes, and mostly consists of data requests and replies. This paper introduces Reactive Circuits, a NoC design that takes advantage of this reactive nature of traffic. This novel design leverages the predictable network traffic behaviour to dynamically build virtual circuits for replies. We evaluate it in a homogeneous chip multiprocessor (CMP) with 16 and 64 cores connected by a mesh.[1]

Analysing the coherence protocol in a standard wormhole 4-stage pipeline router, we note that the request-reply pattern dominates over the rest. Table 1 shows the relative amount of messages travelling through the network, classified into requests and reply types (detailed information about the messages generated by the coherence protocol can be found in Table 3). More than half of the messages are a reply to another message and, therefore, we know their source and destination before the message is injected into the network. With this information, routers can reserve in advance crossbar path and output virtual channel, removing those stages from the critical path of the router pipeline. To hide the circuit reservation latency, we use the requests to make the reservation at every router in the path. We also observed that the network is lightly loaded (nodes inject, in average, less than four flits every 100 cycles), suggesting it will be feasible to keep resources busy for long periods of time. Nevertheless, we also include a version that optimistically calculates when the circuit will be needed and reserves only that timeslot.

Reactive Circuits aims to improve state-of-the-art low-latency NoCs by making the following contributions:

- Implementing a circuit reservation technique with several versions, removing routing and arbitration latency from the router pipeline, and significantly reducing network latency.

[1] A short version of this article was published in [2]. The new contributions are: more comprehensive state of the art review and detailed baseline system architecture description, new diagrams to illustrate the mechanism, new reactive circuit configurations (timed circuits, timed circuits with slack, timed circuits with slack and delay, and postponed timed circuits), ideal circuit reservation included for comparison, detailed analysis of circuits that can or cannot be built and used, network latency results, and performance results for each application for the best configuration run in 64 cores.

Table 1: Percentage of messages that traverse the network (average for all benchmarks executed in a 64-core chip).

Requests	**Replies**		
47.0%	53.0%	**L2_Replies** Data from L2 to L1	22.6%
		L1_DATA_ACK L2 acknowledges data reception	23.0%
		L2_WB_ACK L1 acknowledges write-back reception	4.7%
		L1_INV_ACK Invalidation acknowledgement	1.1%
		MEMORY Data from main memory	0.9%
		L1_TO_L1. Data from L1 to L1	0.7%

- Reserving the circuit on-demand, but hiding the reservation latency with the data request and without any extra circuit setup message or setup network.
- Removing unnecessary buffering from the virtual channel reserved for circuit construction reducing router area and static power.
- Eliminating some acknowledgement messages that are used to guarantee coherence but are no longer needed when data travels through a reserved circuit.

We simulate the mechanism with 16 and 64-core chips using a full-system simulator with realistic workloads. Reactive Circuits reduce network latency, static power and router area. These results emphasize the importance of considering the system as a whole and studying how all the elements interact with each other [3, 1].

The rest of this document is organized as follows: Section 2 presents the state of the art; Section 3 describes the system architecture; Section 4

explains the Reactive Circuits mechanism; Section 5 analyses the simulation results and Section 6 concludes the paper.

2. State of the art

Several works have proposed hybrid packet-circuit switching techniques to speed up certain messages. Some proposals suggest separate networks for packet and circuit switched messages: Palumbo *et al.* determine if messages will use the packet or the circuit switched channels depending on their size [4]; Duato *et al.* decide at compilation time whether a circuit between two nodes should be established based on expected communication patterns [5]; Abousamra *et al.* use the requests to reserve circuits for the replies based on estimates of circuit utilization times [6]; Abousamra *et al.* send a circuit reservation request as soon as a cache hit is detected, though this may not be enough to completely hide the circuit reservation latency [7]. Other authors implement a single network that supports both packet and circuit switching: Enright *et al.* build circuits on demand and undo them when they conflict with another circuit [8]; Abousamra *et al.* configure circuits periodically based on online communication statistics [9]; Kline *et al.* reserve circuits on demand so that flits can traverse multiple hops in a single cycle [10]; Mazloumi *et al.* reserve a circuit with the request and activate it with a probe message when the reply is ready [11]; Liu *et al.* speed up circuit setup in TDM NoCs by sending parallel probes [12].

A different technique preallocates resources in advance to allow faster data transmission, either sending control flits through specialized networks or with tokens that inform neighbouring nodes about their buffer availability [13, 14, 15].

Most of those mechanisms establish circuits between nodes using dedicated networks or links [8, 10, 11, 12] or at least need to send specific setup messages [4, 5, 7], and many of them need to wait for the circuit setup delay [14, 12]. One of them introduces complexity at the network interfaces by forcing them to keep communication statistics [9]. The proposals most similar to our Reactive Circuits mechanism are [6, 11], using the request to reserve circuits for the reply, but they do not go as far as removing buffers to reduce router static power and area or eliminating unnecessary coherence messages. [6] also does an estimation of the time when the circuit will be needed, but does not use it to avoid circuit conflicts.

Another common approach to reduce network latency involves routers that speculate by using paths without prior reservation, which only work if there is no contention [16, 17, 18, 19]. These routers are more complex and may require reduced network frequency or result in energy and performance penalties when the implemented shortcuts cannot be used. There are also several proposals that radically modify the routers to eliminate all buffers and reduce per-hop latency to one or two cycles, but suffer big penalties with any level of congestion [20, 21].

Many of the publications we have mentioned do not perform full-system simulations with real traffic, and therefore, are unable to capture the effect of realistic traffic patterns on their proposals [4, 5, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21].

The Reactive Circuits proposed in this paper uses some of the same concepts already introduced by other publications, but combines them to implement an optimized mechanism that does not require extra networks, additional management messages, gathering statistics, or modifying the coherence protocol. We leverage the memory hierarchy behaviour to efficiently reserve network resources in advance with minimal changes to the routers, and completely hiding circuit reservation delay.

3. System architecture

This work focuses on a homogeneous CMP where each tile is composed of a single-threaded core with private first level cache and a bank of the shared second-level cache, both connected directly to the router. Some tiles in the edges of the chip also include a memory controller. Figure 1 depicts the block diagram of the chip and a tile with memory controller. It also includes the connections between the elements in the tile and the router. Table 2 presents the key parameters of the architecture and Table 3 details the messages exchanged by the coherence protocol. It is a MESI protocol that allows direct data transfer between L1 caches, as opposed to a simpler version that always forced to use the L2 as an intermediary. To model the architecture we based our design on other systems with similar characteristics, both from academia [22, 23, 24] and industry (Tilera's TILE*Pro*64 [25], Intel Xeon Phi [26], and Intel 48-core processor [27]).

The baseline NoC that connects all tiles is built as a mesh with simple 4-stage routers, XY routing and wormhole flow control. Wormhole flow control was chosen over virtual cut-through and store-and-forward because it

minimizes latency and power consumption. Store-and-forward requires all the flits of a packet to be received at a router before they can be sent to the following router, thus increasing latency. Virtual cut-through starts sending flits forward before the whole packet has arrived, but still must allocate resources for the whole packet. Our circuit reservation mechanism can be used with any of those flow-control strategies, and the obtained speedups would higher than with wormhole, which is an more efficient baseline. Table 4 shows the detailed configuration of the baseline NoC and Figure 2 depicts the router architecture.

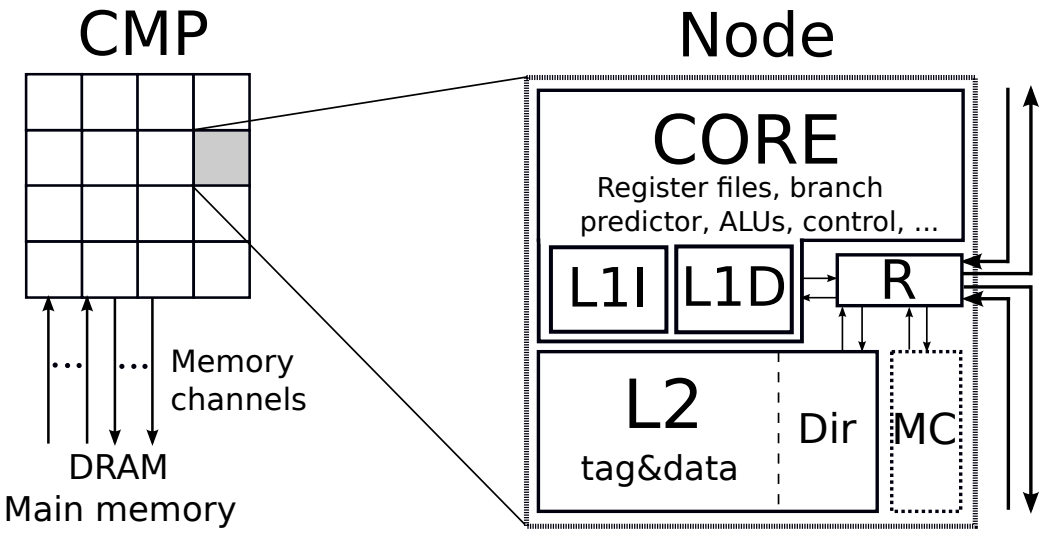


Figure 1: Block diagram including a chip and the components of a tile. MC stands for memory controller, R is the router, and Dir is the directory, which is included in the L2 cache bank. This example router has two input and two output ports connected to neighbouring tiles.

4. Setup, Operation, and Release of Reactive Circuits

This section describes the details of the Reactive Circuits mechanism for each of the implemented versions: reserving, using, and undoing fragmented and complete circuits, sharing circuits, eliminating coherence messages, and building timed circuits.

Table 2: Main characteristics of the chip multiprocessor.

Processors	16 y 64, Ultrasparc III Plus, in order, IPC 1, single-threaded, 2GHz frequency
Coherence	Directory based, MESI, directory distributed in the L2 banks
Consistency	Sequential
L1 cache	32KB data and instruction caches, 4-way assoc, 2-cycle hit, 64B lines, private, pseudo-LRU replacement
L2 cache	Distributed, 1 bank/node, 1MB/bank, 16-way assoc, 7-cycle hit, 64B lines, shared, inclusive, pseudo-LRU replacement
Memory	4 memory controllers distributed in the edges of the chip (for both 16 and 64-node chips), 160-cycle latency

Table 3: Messages generated by the coherence protocol.

Event	**Sequence of messages**
L1 miss	1º Request from L1 to the corresponding L2 bank 2º L2_Replies: Data reply from L2 to L1 3º L1_DATA_ACK: ACK from L1 to the L2 bank
L1 miss, another L1 owns the data exclusively	1º Request from L1 to the corresponding L2 bank 2º L2 forwards the request to L1 owner 3º L1_To_L1: L1 owner sends data to L1 requestor 4º L1_DATA_ACK: ACK from L1 requestor to the L2 bank
Invalidation (write or L2 replacement)	1º Invalidation from L2 to L1 sharers 2º L1_INV_ACK: ACK from L1s to the L2 bank
L1 replacement	1º Replacement data from L1 to the corresponding L2 bank 2º L2_WB_ACK: ACK from the L2 bank to L1
L2 miss	1º Request from L2 bank to the corresponding memory controller 2º MEMORY: Data from the memory controller to L2 bank
L2 replacement	1º Replacement data from L2 bank to the corresponding memory controller 2º MEMORY: ACK from the memory controller to L2 bank

Table 4: Main characteristics of the baseline network on chip.

General	2 virtual networks (VN), requests and replies 2 virtual channels (VC) per VN
Routers	4 stages: routing and input buffering, VC allocation, switch allocation, and switch traversal round-robin 2-phase VC/switch allocators 5-flit buffers per VC, enough to store a whole message
Links	16B flits, 1-cycle latency

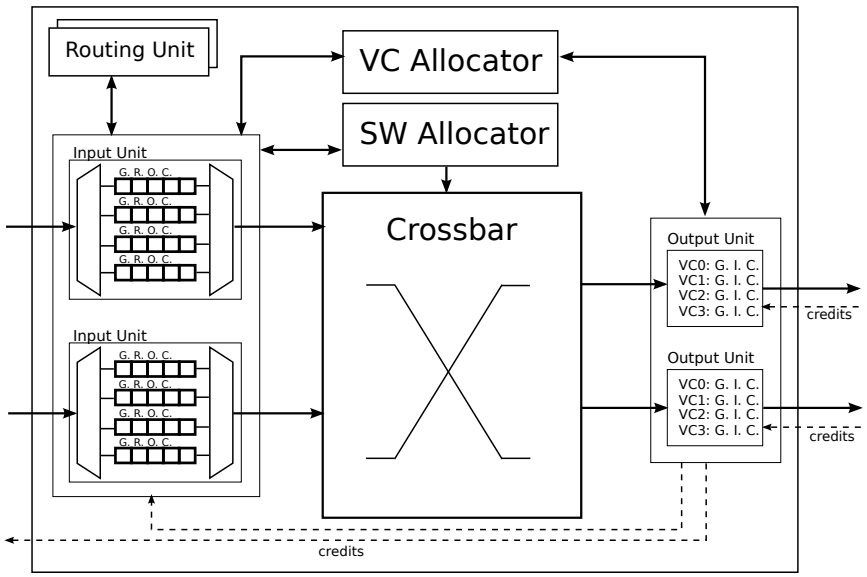


Figure 2: Architecture of the baseline router. VCs at the input units store global state (G), route (R), output VC (O) and credit count (C). At the output units, they store global state (G), input VC (I) and credit count (C).

4.1. Reserving reactive circuits

As discussed before, when a request reaches its destination, we already know that a reply is going to be sent back to the source, based on the patterns introduced in Table 3. A previous approach consists of sending the head of the message in advance to reserve the resources along the way, building a circuit for the data in parallel with the L2 access [7]. However, in our case the L2 hit access is too fast (7 cycles) compared with the average time needed to set up the circuit (19 cycles in a 16-core chip and 59 in a 64core system). We overcome this problem by reserving reactive circuit for the reply as its request travels towards the destination. Using dimension order routing (DOR), request and reply follow disjoint paths because both messages travel first in the horizontal direction, and then in the vertical one. In consequence, we start by modifying the DOR algorithm so that requests and replies use XY and YX routing, respectively, so the path to and from the destination match. This change does not generate deadlocks because different message types use different virtual networks.

Requests go through the four original stages of the router (see Table 4). In parallel with VC allocation, the reactive circuit is built for the reply. During that process, the necessary information to identify the circuit is stored in the router (requestor identifier and cache line address). Since we have two VCs for replies, we dedicate one to circuits and leave the other for replies that do not have a circuit. This way, Reactive Circuit routers will support packet and circuit switching simultaneously. Information of the circuit is also stored in the network interface where the circuit starts.

Out of the message types in Tables 1 and 3, reactive circuits are built for data sent from L2 to L1 (L2_Replies), replacement acknowledgements (L1_WB_ACK), and main memory replies (MEMORY), which account for 53.2% of all reply messages. Invalidation acknowledgements (L1_INV_ACK) and direct data transfers between L1 caches (L1_TO_L1) are a very small percentage of the reply messages (only 1.8%). L1_DATA_ACK messages are replies sent from an L1 to an L2 after a request-reply communication to confirm the reception of the DATA. These messages are essential to maintain coherence as they guarantee the L2 that the data has been received and prevent race conditions, and are used both in academia [28, 29] and industry, e.g. AMD Opteron [30]. L1_DATA_ACK messages do not follow the same path as the request and reply between L1 and L2: the request follows the XY path from the L1 to the L2 and the reply follows the YX path from L2 to L1 (so it goes through the same routers as the request), but the ACK is a

reply that follows the YX path from L1 to L2. Therefore, it is not possible to use a previous message to build a circuit for them.

The following sections describe specific details of the implementation and introduce the different versions of the Reactive Circuits mechanism.

4.2. Fragmented versus complete circuits

When trying to build a circuit at a router, the necessary resources might not be available. In this situation, there are two design alternatives:

- Allow *fragmented circuits*, keeping the partial path reserved, and attempt to reserve the rest of the path after the next hop.
- Support only *complete circuits*, so that any lack of resources will force us to undo the previous reservations.

With *fragmented circuits*, we need to assure messages can always be stored in the router in case their circuit has not been completely built. As we already mentioned, we start by dedicating one VC for replies without a circuit, and the other one for replies with circuit. In the baseline NoC, VCs are not heavily used and are rarely blocked. However, keeping VCs reserved for a longer period of time has a negative effect: there may not be enough resources to exploit the full potential of the proposal. Therefore, with fragmented circuits we include an additional VC to increase the number of simultaneous circuits, ending up with a total of three VCs in the reply virtual network. This extra VC will mean an increment in router power and area.

Building only *complete circuits* allows us to implement many simplifications in the router. We guarantee that a message with a circuit has all the resources it needs from source to destination. Hence, it will never get blocked in the network. This has two beneficial effects: first, it allows us to remove the buffer storage of the VC dedicated for circuits reducing the router area; second, we can build as many circuits as we want for that VC in every input port because flits will just go through the router without stopping.

All the complete circuits in the same input port of a router must have the same source to avoid conflicts: two circuits with different input ports and the same output port cannot be built at the same time on a router. This is because if two flits arrived at the same time wanting to use those circuits, one would have to be dropped because both of them would not be able to leave through the same output port in the same cycle. We have experimentally explored the best number of simultaneous circuits built per input and set

it to five. This number reduces the probability of failing to build a circuit due to lack of free storage for circuit information, but it is small enough to minimize area and power consumption, as we will demonstrate in the evaluation section. Figure 3 presents the modified router that implements the reservation of complete circuits.

To clarify the difference between the two alternatives, we show an example of how circuits are built in Figure 4. To simplify the example, we assume there is one single VC dedicated to circuits. In both cases, there is a blue circuit already built from L2A to L1A, and a new request tries to build a new circuit (green) from L2B to L1B. The request builds the circuit as it is traversing the network, so the circuit is built starting from its final router (L1B) and ending in its first router (L2B), in the opposite direction of the replies that will use it. In Figure 4a, there is one hop in the network where a conflict is detected (in the router marked as R2), but the fragmented circuit can be built in all the other routers. In Figure 4b we see that the situation is very different with complete circuits. When the request tries to build the circuit in router R1, it detects there is already another circuit using the needed input port (East port). That circuit that is already completely reserved has a different source than the one we are trying to build now, which means that at some point we will need two circuits with different inputs and the same output in a router (in router R2). Therefore, this circuit cannot be reserved in all the routers and, since this mechanism only supports complete circuits, the successful reservations in the downstream routers have to be undone.

Both fragmented and complete circuits can be implemented with any deterministic routing, as long as we can force requests and replies to go through the same routers. With adaptive routing, only the complete circuits version of the mechanism would work. However, we have not explored an adaptive alternative because the router overheads to avoid deadlock are high and would result in longer base latency.

4.3. Using the circuits

When a reply arrives at a router, it checks if there is a circuit built for it. In that case, it can go straight through the crossbar leaving the router in just one cycle. When the tail flit of the message leaves the router, it frees the circuit resources by clearing the B bit. With fragmented circuits, a message that had a circuit might arrive at a router where there is no built circuit.

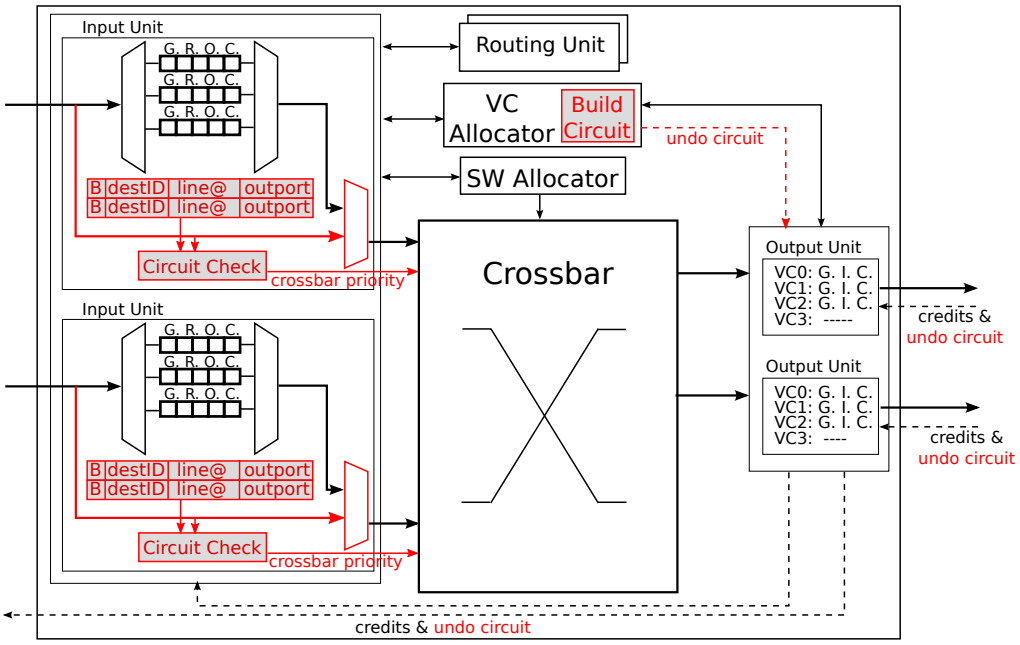
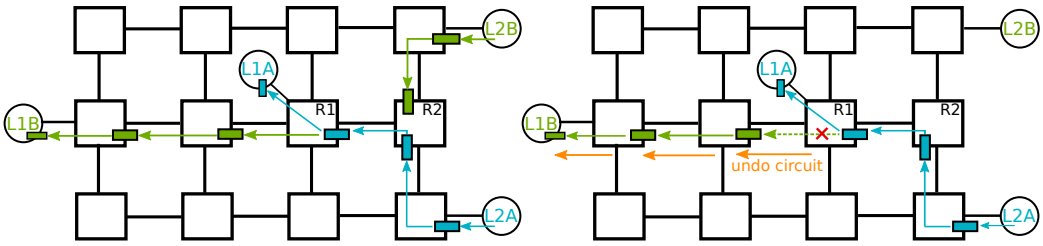


Figure 3: Architecture of the router that reserves complete Reactive Circuits. The modifications with respect to the baseline router are highlighted. They include "Circuit Check" logic at the Input Units and a "Build Circuit" module in the VC allocator. In this drawing, two simultaneous circuits can be built per input port. VCs at the Input Units store global state (G), route (R), output VC (O) and credit count (C). Circuit information includes built-circuit bit (B), destination identifier (destID), cache line address (block@) and output port (outport). Credits may carry undo-circuit information. The Output Units store global state (G), input VC (I) and credit count (C).



(a) Fragmented circuits (b) Complete circuits

Figure 4: Example of how circuits are built with fragmented and complete circuits, using always one single VC for circuits. In both cases, the blue circuit has already been built by a request going from L1A to L2A. Then another request has tried to build the green circuit from L2B to L1B.

When that happens, it will just be stored in the VC and go through the usual four stages of the router.

Even when there is a circuit built at a router, the ports and links involved can still be used by other messages. The crossbar prioritises messages with a circuit, but it grants access to the other virtual channels when the circuit is not used.

4.4. Undoing circuits before they are used

We must undo a circuit before it gets used under the following situations:

- The coherence protocol lets an L2 cache bank forward a request to an L1 that owns a cache line exclusively, who will supply the data directly. Therefore, the circuit built between the requestor L1 and the L2 bank will never be used and should be undone.
- When we try to build a complete circuit and reach a router where resources are not available, we have to undo the section of the circuit we had successfully built so far (see Figure 4b).

In both those situations, we undo the circuit with a simple and efficient technique: we send the data of the circuit to be undone towards the circuit destination using credits. If a credit had to be sent at the same time to free a buffer, we piggyback the information; otherwise, we send a specific credit.

We also considered undoing circuits when an L2 miss occurs, because resources will be held for a long time while the request goes to main memory. However, simulation results show better performance if we keep them built.

4.5. Reusing complete circuits

In the previous sections, circuits were specifically built for a message and used only by that message. We go a step further to improve our mechanism and try to find other messages that can reuse the circuits. When a reply that does not have a built circuit is about to leave the network interface, it checks if there is any circuit starting at that NI that it could use to get closer to its destination. In that case, the message becomes a *scrounger* message that uses the circuit to reach an intermediate destination. At that point, the network interface will forward the message by re-injecting into the network that it can arrive at its final destination.

Note that we can only apply this method with complete circuits because there are no buffer guarantees for two messages using the same fragmented circuit.

4.6. Eliminating coherence messages

Studying the coherence protocol while designing the NoC has allowed us to notice a rewarding effect of our reactive complete circuits. We already mentioned that we cannot build a circuit for the L1_DATA_ACK reply messages that are sent from L1 to L2 after the L2_Reply (see Section 4.1). Since the NoC does not guarantee message ordering, this L2_DATA_ACK avoids a situation where the L1 received an invalidation or a forwarded request for data it had not received yet. However, if the L2_Reply uses a complete circuit to get to the L1 requestor, we are sure the data, travelling at a speed of 2 cycles per hop and never blocking, will arrive before any other message sent from L2 to L1 afterwards. Therefore, we can acknowledge the data reception to the L2 without the need to wait for the L1_DATA_ACK message. With this simple observation, we can omit those messages to reduce contention in the network and energy consumption. On top of that, other requests waiting to access the same cache line will reduce their waiting time since the L2 cache line will not be blocked while the L2_Reply and L1_DATA_ACK messages are exchanged.

4.7. Timed reservation of complete circuits

Having complete circuits is the most beneficial option to reduce router power and area. However, circuits cannot be built when there is a conflict, which means there cannot be two circuits with different input ports and same output port built simultaneously in a router. If two flits arrived at the router at the same time wanting to leave through the same port, one would be forced

to wait and we would not be able to store it because we have removed the buffers for that virtual channel. However, given the light load of the network, it is very improbable for those two flits to arrive at the same time and create a real conflict.

Therefore, circuits cannot be built due to the possibility of a collision that may not actually happen. To avoid that, we implement timed circuit reservation: we optimistically calculate when the reply will go through the router, and reserve the circuit only for those cycles. This way, the channel is not busy from the moment it is reserved until it is used, it will only be busy for a short time interval. The time will be calculated using the number hops between the current router and the destination, the hop latency for the request (five cycles/hop) and for the reply (two cycles/hop), and the cache hit latency [6]. It will then be stored in two counters where we will annotate the cycles until the circuit reservation starts and finishes, and that will be decreased every cycle. Abousamra *et al.* also calculate the expected reply arrival time but use it only order the circuit reservations, stating that timed based reservations are impractical due to unforeseen delays. We address this issue by enhancing the basic idea with three variations of complete timed circuits, as explained below.

Now, circuits with different input port and same output port can be built at the same time, as long as they use non-conflicting time slots. When a reply is going to be sent, it can only use its circuit if it is within the optimistic timing estimation. Otherwise (for example in case of a cache miss), the circuit will be undone and the reply will need to go through all the stages of the router. Figure 5 describes how timed circuits are reserved and includes three variations designed to increase the flexibility:

1. Reserve the circuits with *slack*. Instead of reserving the exact number of cycles the reply will need, we give the option to reserve more cycles to be able to accommodate delays due to failed arbitrations of the request and extra cache delays.
2. Allow reserving the circuits with *delay*. If the time slot the circuit needs has already been occupied, we try to reserve the circuit for some cycles later. The reply may need to wait for its time slot before being sent, but it will reach its destination faster by using the circuit. Note that this version must be combined with the previous one: we need to reserve the timeslot with a slack so that we can introduce a delay and still be on time for the reservations already made in previous routers.

3. To have the flexibility of the slack without reserving the circuit for a longer period of time (which increases the probability of conflicts), we reserve *postponed* circuits. In this case, we reserve the circuit for the exact number of cycles it needs, but for a later time. This will increase the number of circuits that can be built and used, but all the replies will need to wait for the circuit, even if the request was not delayed and they were ready before.

In the three versions, the number of cycles of slack, delay or postponement is proportional to the path length, introduced as number of cycles per hop.

4.8. Ideal circuit reservation

We consider an ideal version of the mechanism that will successfully reserve and use all the circuits. This version is not a feasible design due to the increased area and power consumption, and the inclusion of logic that wouldn't fit in a single cycle, but we include it as an upper bound for performance comparison. It consists of keeping the buffers and reserving all the circuits, without caring about conflicts or timing, and without a limit in number of circuits per input port. Then, all the replies will use their circuit to reach their destination. At every hop, the router needs to check if there are two conflicting flits using circuits, and in that case, prioritise one of them and keep the other in the buffer. That is done in a single cycle, as well as checking if the circuit has credits for the next hop before forwarding the flit. We would not be able to implement this in a real system, but all the replies will use circuits and suffer only small delays if there are collisions, which will give us the best performance Reactive Circuits can offer and will be useful for reference.

5. Evaluation

This section presents the simulation methodology and the main results for the Reactive Circuits techniques, including power, area, and performance.

5.1. Simulation framework and workloads

We use Simics [31], GEMS [32] and an extended version of Garnet [33]. We carefully model all the components of the chip and perform full system simulation with single-thread cores and directory-based coherence. To get the timing, area and energy expended by the network we use DSENT, a

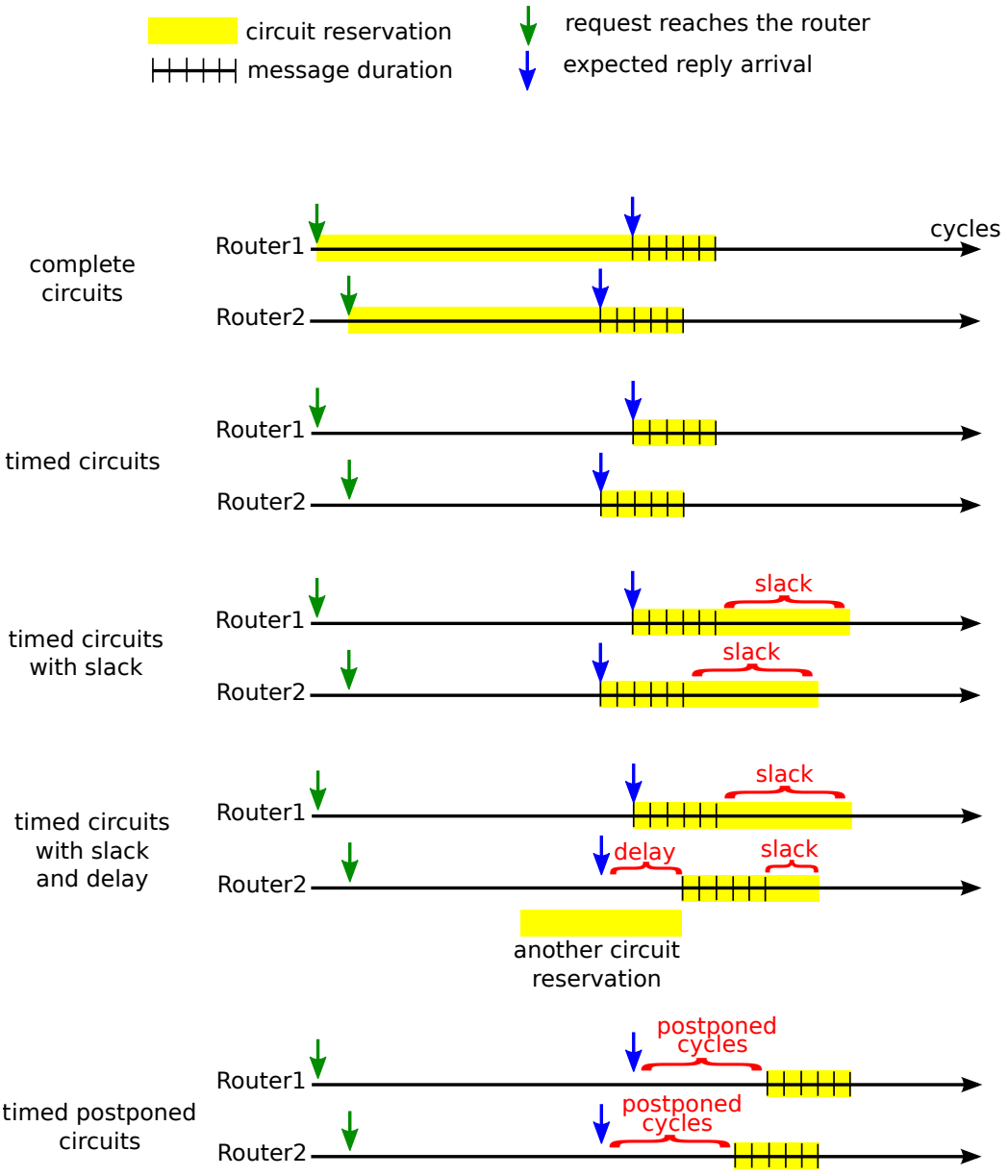


Figure 5: Diagram for reactive circuit reservation in the four variants of complete timed circuits. Basic complete circuits are also included for comparison. In each configuration, the construction of the circuit is shown for two consecutive routers.

state-of-the-art circuit modelling tool [34]. We assume 32 nm technology and run at 2 GHz frequency.

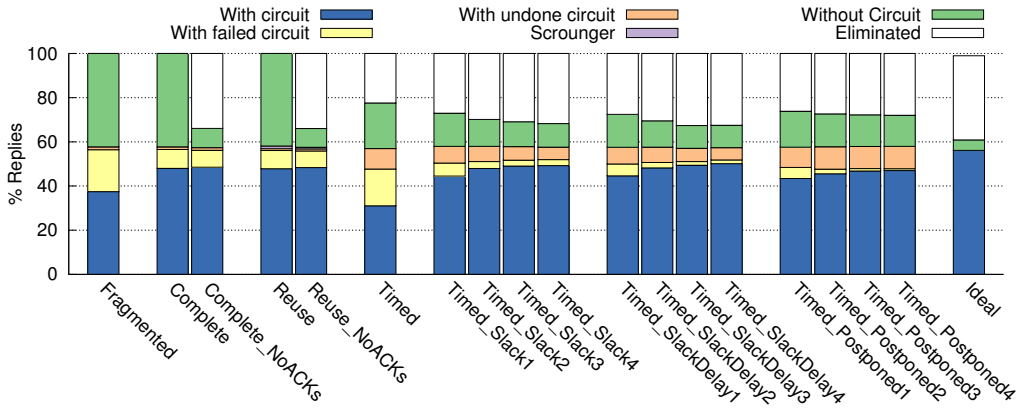
CMPs can execute parallel applications to reduce execution time, or multiprogrammed workloads (execution of independent applications on each core) to increase throughput. We use parallel applications from PARSEC [35] (`blackscholes`, `bodytrack`, `canneal`, `dedup`, `ferret`, `fluidanimate`, `raytrace`, `swaptions`, `vips`, and `x264`) and SPLASH2 [36] with scaled inputs from PARSEC 3.0 (`barnes`, `cholesky`, `fft`, `lu_cb`, `lu_ncb`, `ocean_cp`, `ocean_ncp`, `radiosity`, `volrend`, `water_nsquared`, and `water_spatial`). We run the applications with 16 and 64 threads in the 16 and 64-core chips, respectively, and simulate the whole parallel region.

For the multiprogrammed workloads, we choose 16 applications with a large working set from the SPEC CPU 2006 suite [37] and bind each application to a different core. To build the workloads for the 16-core chip, we randomly distribute the applications to build a mix. For the 64-core chip, we use each application four times, and again build a random mix. To perform the evaluation, we warm up the caches for 200 million cycles and simulate for 500 million cycles.

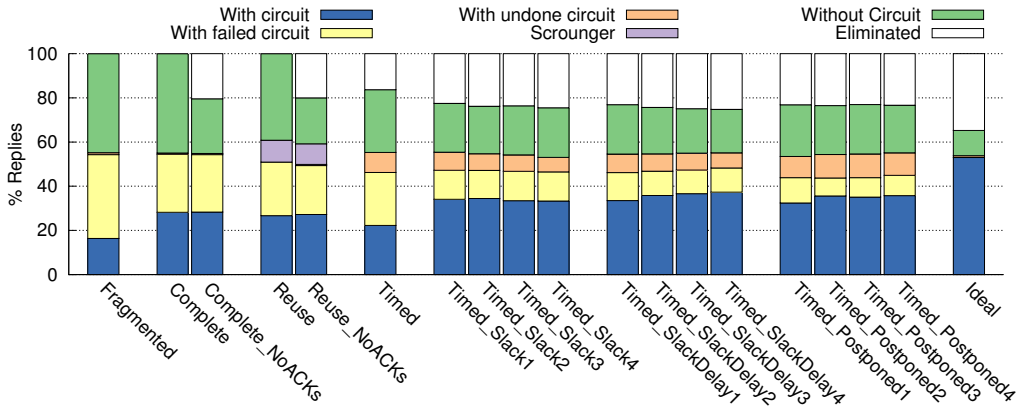
5.2. Construction and use of Reactive Circuits

We analyse how effective each version of our mechanism is in building and using circuits. Figure 6 presents the percentage of replies that travel on a circuit, with a failed circuit (could not be completely built), with an undone circuit (it was completely built but had to be undone), that travel on a circuit built for another message (scrounger messages), that were not eligible for a circuit, and that were eliminated (removed L1_DATA_ACKs due to successful L2 to L1 circuits). It includes every circuit-building configuration tested in 16 (Figure 6a) and 64-core chips (Figure 6b), and we present the average of all the parallel applications and the multiprogrammed mix.

The first bar of the graph corresponds to fragmented circuits. In this case, the failed circuits are those that could not be completely built, but replies using them will still have sections of their path with a built circuit. As we already anticipated in Section 4.4, there are some cases when a built circuit will not be used due to the behaviour of the coherence protocol (when the L2 bank forwards the request to the L1 owner). However, this is a very small percentage of the total of replies. Apart from that, there are more than 40% of replies that cannot benefit from the mechanism because they are not associated with a request that can reserve the circuit.



(a) 16 cores



(b) 64 cores

Figure 6: Percentage of replies that travel on a circuit, with a failed circuit (could not be completely built), with an undone circuit (it was built but had to be undone), that were scrounger messages, that were not eligible for a circuit, and that were eliminated, for every circuit-building configuration tested.

The rest of the bars are different versions of complete reactive circuits. We detect that in this case we can reserve more successful circuits (blue section of the bars). This is because fragmented circuits have to guarantee a buffer for all the replies, which forces us to set a low maximum of circuits per port (two in this case). On the other hand, replies with complete circuits will never block, so they do not need a buffer. This allows us to reserve more simultaneous circuits per input port (five in our case), and almost all circuit failures come exclusively from output port conflicts (when we would need two circuits from different input ports to the same output port). Removing coherence messages (*NoAck*) has a significant impact by eliminating 20-30% of the replies. On the other hand, reusing circuits has only some impact on the 64-core configuration, because there are more circuits built on the network, and, therefore, a higher probability for scrounger replies to find a suitable circuit.

We then present results for basic timed circuits and three additional versions, always removing the non-necessary coherence messages. The three versions correspond to the ones introduced in Section 4.7 (Slack_, SlackDelay_, and Postponed_, where "_" is the number of cycles per hop). They are all simulated with different values for the slack, which is introduced as number of cycles per hop in the path. This way, the slack automatically adapts to the path length. In the basic timed version, we notice that there are more failed circuits than in the simple complete circuits scheme, especially in the 16-core system. This is because the strict timing restrictions cause the circuit to fail as soon as the request suffers any delay (loses any VC or switch arbitration), the optimistic timing calculation performed for the reply does not stand any more after that. We clearly see in the figure how the number of successful circuits rapidly increases as we introduce slack, effectively solving the problem. However, especially with 64 cores, we realize that increasing the slack does not necessarily allow more circuits to be built. This is because there is a trade-off in the number of cycles of slack we reserve: with a small slack, circuits fail because the timing cannot be met after small delays; on the other hand, higher slacks give more room for delays, but reserve circuits for longer periods of time, making it more likely to have conflicts in output ports.

Apart from that, we notice a negative effect in all the timed circuits: the amount of circuits that get completely built but have to be undone without being used significantly increases. In the versions of the mechanism without timing, this was only caused by a pattern in the coherence protocol that

happened sporadically (the L2 bank forwarding the request to the L1 owner). However, with timed circuits a reply must leave the network interface exactly within the reserved timeslot; otherwise, the circuit must be undone and the reply has to follow traditional router pipeline. This unpleasant situation happens due to unpredictable delays in the caches, mostly because requests are blocked in busy cache lines waiting for acknowledgements.

The last bar with the ideal circuit construction has been included for comparison. In the 16-core chip, our mechanism achieves results very close to the ideal, while the 64-core chip cannot exploit the mechanism to its fullest potential. Comparing Figures 6a and 6b, we notice that it is more complicated to build circuits with a larger chip, making the scalability of the mechanism a concern. This is due to the longer paths messages need to follow and the increased amount of traffic, which generate more conflicts and cause circuits to fail. This means that less replies will be able to reduce their latency and that more replies will need to use the same non-circuit VC, thus increasing latency. With the basic version of complete circuits for 64 cores, only about 25% of replies use a circuit, the remaining 75% must use the other VC, thus increasing congestion. This situation is however improved by two optimizations: removing acknowledgements reduces the amount of replies using the non-circuit VC down to about 50%; timed circuits increase the amount of replies that can use a circuit to about 40%, and in turn, also increases the number of acknowledgements that can be removed. With all these optimizations, there are less than 40% of replies contending for the non-circuit VC. Assuming that in the baseline configuration both VCs would be used equally (50% of replies in each VC), with the most optimized reactive circuits version we are actually reducing the load of that VC and maintaining the benefits of Reactive Circuits. We expect the effect of those optimizations to be even more relevant with bigger chips.

In the complete circuits versions, we can reserve several circuits per input port. As we explained in Section 4.2, we experimentally choose the number of simultaneous circuits to be big enough to reduce failed circuits due to lack of storage but small enough to minimize area and power. As an example, Table 5 presents the number of simultaneous circuits built for the complete circuits version with eliminated coherence messages in a 64-core chip. The table includes the percentage of circuit reservations at routers that correspond to the first, second, third, fourth, and fifth reservation in the same input. We notice that it is much more common to reserve the first circuit at an input port that it is to reserve the second or third. Nevertheless, the

Table 5: Percentage of circuit reservations in all routers that correspond to the first, second, third, fourth, and fifth reservation in that input. The percentage of failed circuits is also included.

Avg. circuit reservations in routers	**1st circuit**	**2nd circuit**	**3rd circuit**	**4th circuit**	**5th circuit**	**failed**
	48%	24%	7%	6%	6%	9%

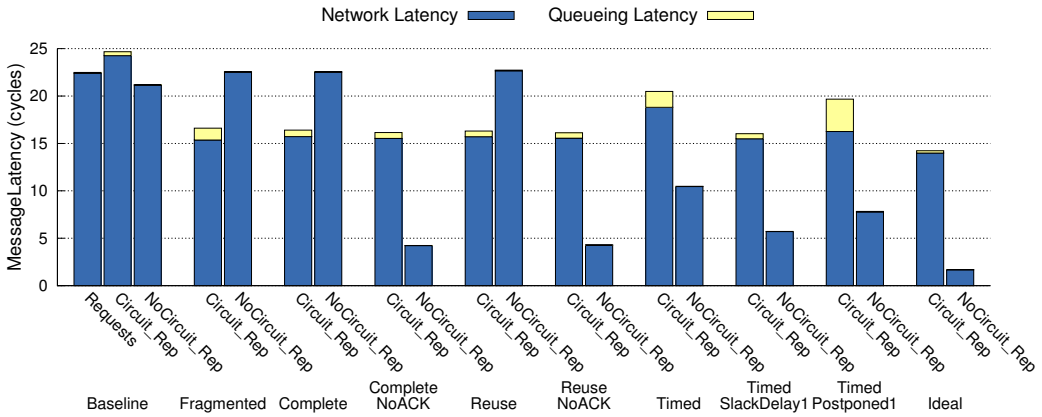
storage for all the five circuits is used and it leaves a small percentage of failed circuits due to lack of storage.

5.3. Network Latency

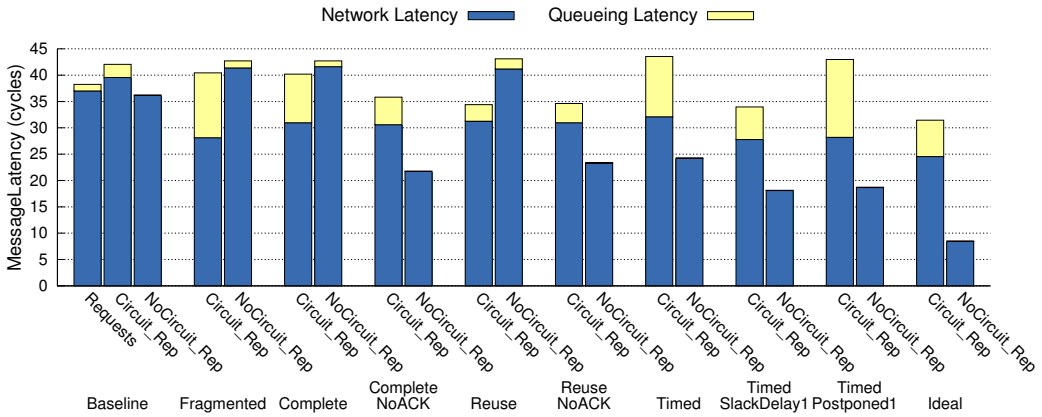
Figure 7 shows how the circuit construction affects message latency depending on the type of message: requests, replies eligible for circuit construction (Circuit_Rep), and replies for which we cannot build a circuit (NoCircuit_Rep). We include the baseline and ideal configurations, and the most relevant versions of the Reactive Circuits mechanism. Since the latency of requests does not change in any of those versions, we show it only in the baseline experiment. In each bar we distinguish between network latency (cycles each message spends in the network) and queueing latency (cycles before the message can enter the network). In the baseline configuration we see that the replies eligible for construction have higher latency than the requests, which is because most of them have five flits instead of one; replies not eligible for circuits are normally acknowledgements composed of a single flit.

When we build circuits for the replies, either fragmented or complete circuits, the network latency is significantly reduced. The highest savings are obtained with the basic complete circuits, reusing circuits, and timed circuits with slack and delay, always removing unnecessary acknowledgements. To make a fair comparison, we have considered the latency of the eliminated coherence messages to be zero. In the configurations where we remove those messages, we notice a dramatic drop in the latency of replies that are not eligible for circuits.

The timed circuits without any slack do not reduce network latency as much as the other options because, as we already showed in Section 5.2, not many circuits can be successfully built. We include two of the optimized versions of timed circuits: one with slack and delay, which significantly reduces the latency, and one with postponed circuits. The latter was implemented



(a) 16 cores



(b) 64 cores

Figure 7: Message latency for different types of messages (requests, replies eligible for circuit construction (Circuit_Rep), and replies for which we cannot build a circuit (NoCircuit_Rep)) and Reactive Circuit versions, averaging the result from the parallel programs and the multiprogrammed mix.

to increase the number of built circuits, but this was done by forcing a delay for every reply. Even though we can reserve many circuits, the forced delay has a negative impact on network latency. In fact, this option will not result in performance or energy improvements, so we will not include it in the following sections.

We notice that Reactive Circuits have a negative effect, especially in the 64-core chip: the queueing latency increases significantly, as well as the network latency for non-circuit messages. This is because virtual channels are now dedicated to each traffic type (circuit or non-circuit), so we eliminate their use as virtual lanes to reduce congestion, thus increasing the latency for non-circuit messages. Luckily, we can partially solve it by eliminating the unnecessary coherence messages, which lightens the load of the non-circuit VC.

5.4. Router Area and Network Energy

Table 6 presents the savings in router area for each version of the mechanism compared with the baseline router with four VCs. We assume that links are routed over logic, and therefore do not contribute to network area. With fragmented circuits, the area increases by almost 20% because we had to include an extra VC for circuits in order to increase the number of simultaneous circuits, as well as storage for the circuit information. In contrast, with complete circuits we also need to include storage for circuit information but we can eliminate the buffers in the VC dedicated for circuits, which makes the router area decrease by 6%. When enhancing complete circuits with timed reservations, we must also store the circuit timestamps, which cancels the benefit or removing the buffers almost completely. We always remove the buffering from one VC at every port in every router, therefore, these area savings will be maintained when scaling the chip to larger sizes.

These benefits in area, along with the speedup achieved as a result of the network latency reduction, translate into outstanding energy savings. Figure 8 depicts the normalized network energy for the most relevant configurations, including dynamic and static energy for both routers and links. The ideal version is not included because it involves unlimited storage for circuit information. With fragmented circuits, the energy increases the same way the area did. However, for the rest of versions, we substantially reduce the energy. The versions without unnecessary coherence messages involve further improvements due to the reduction in execution time and network utilization. The complete circuits removing the acknowledgements achieve

Table 6: Router area savings in the different versions of the circuit-building mechanism. Negative values correspond to configurations with larger area.

Version	**Area Savings**	
	16 cores	64 cores
Fragmented	-19.28%	-18.96%
Complete	6.21%	5.77%
Complete Timed	3.38%	1.09%

the highest savings, with energy reductions of 15.2% and 20.8% in 16 and 64-core chips, respectively. The effect of the mechanism on the 64-core chip is more relevant because the network has a higher impact on larger systems. Using the different versions of complete Reactive Circuits, we will always save network energy from the elimination of buffers, as long as there is not a degradation in performance that reverses the trend by significantly increasing execution time.

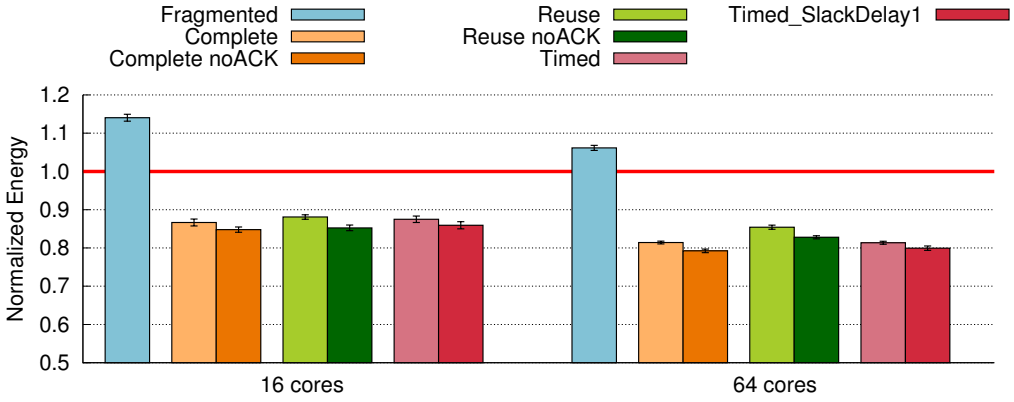


Figure 8: Network energy for the different versions of the Reactive Circuits mechanism normalized to the baseline without circuits. We present the average of all parallel applications and one multiprogrammed workload and include the standard error for every configuration.

5.5. System Performance

Figure 9 presents the average speedup of all parallel applications and the multiprogrammed mix for the most relevant versions of the mechanism. We notice that the speedups are not very large, mainly because the network is lightly loaded, which limits the effect of network latency on overall performance. Other similar proposals do not mention performance in their results, probably because the nice improvement in network latency translates into small performance improvements, like in our case. The speedup achieved by our mechanism is very close to the ideal one. Differences among versions are slightly more pronounced in the 64-core chip, where the network has a larger impact. The versions where we eliminate unnecessary coherence messages consistently achieve better results than their counterparts with all coherence messages. The version with the best performance results is the timed circuits with slack and delay, with performance improvements of 4.4% and 6.0% for 16 and 64 cores, respectively. Non-timed complete circuits had larger energy savings than timed circuits even though their speedup is slightly lower (3.8% and 4.8% for 16 and 64 cores) because they do not need to store circuit timestamps.

Figure 9 also includes the standard error for every configuration, which is very small. The margin of error of our results with a confidence level of 95% is always less than 2% for 64 cores and less than 5% for 16 cores [38]. These results point out that, even though performance gains are small, they are consistent across all the simulated applications and statistically significant.

For complete timed circuits with slack and delay in a 64-core chip we present the speedup for each application in Figure 10. We can see that 50% of the simulated applications experience performance gains over 4.5%. There are several applications where the Reactive Circuits mechanism is especially beneficial and experience performance improvements above 10%, while only two applications out of the twenty two experience a very small slowdown (less than 2%).

Under very adverse conditions, with heavy traffic loads, conflicts would be frequent and prevent complete circuits from being built, lowering system performance. However, timed circuits reduce the time circuits keep virtual channels occupied, thus rising the threshold over which the network would be too congested to build circuits and reduce latency.

With the studied chip sizes (16 and 64 cores), all the versions of the mechanism achieve similar speedups. As the chip size increases, paths will be longer and there will be more messages using the network simultaneously.

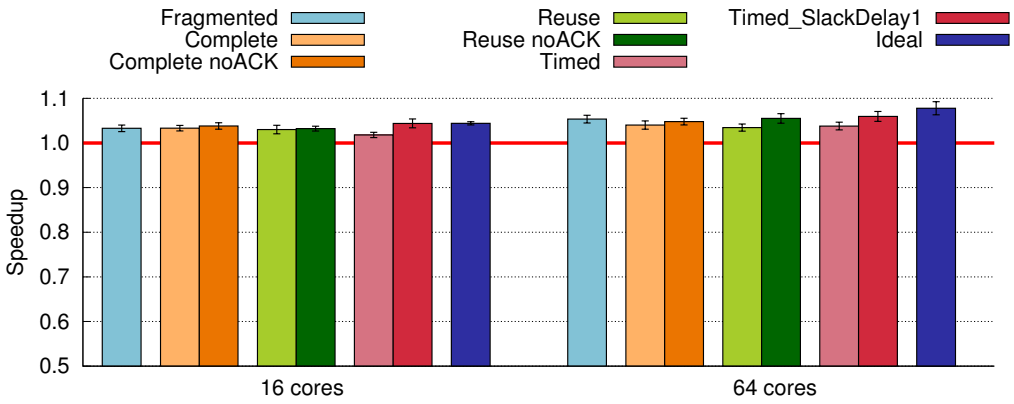


Figure 9: Speedup for the different versions of the circuit-building mechanism with respect to the baseline without circuits. We present the average of all parallel applications and the multiprogrammed workload and include the standard error for every configuration.

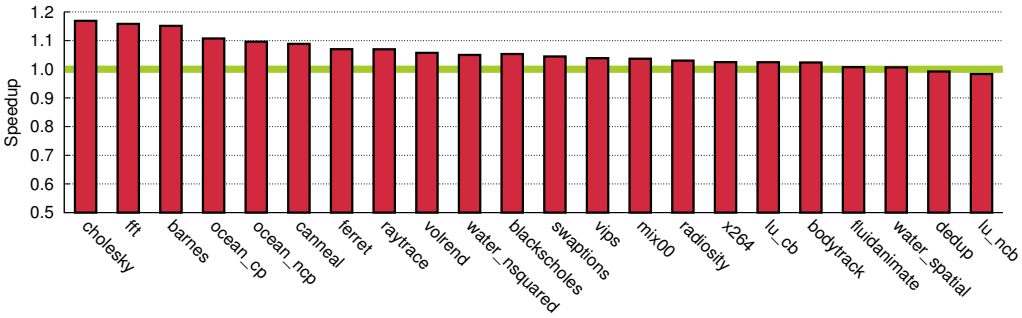


Figure 10: Speedup with respect to the baseline for every parallel application and the multiprogrammed mix for timed reactive circuits with slack and delay of 1 cycle per hop.

This will generate more conflicts and it will be more complicated to build complete circuits. For these reasons, timed circuits will be very useful to guarantee the scalability of the mechanism. They will only keep resources busy for short periods of time, thus reducing conflicts compared with non-timed circuits. Apart from that, it is considered that future systems with hundreds of cores will not be used monolithically to run one single application. Workloads do not offer enough parallelism to run efficiently on such a large number of cores. Therefore, the usage model of near-future networks-on-chip will likely involve partitioning and partition isolation, as it has already been implemented by Tilera with their Multicore Hardwall mechanism [39]. In a partitioned system, Reactive Circuits could be used independently inside each partition, thus eliminating concerns about the need to scale to a larger number of cores.

6. Conclusions

CMPs are composed of multiple nodes connected via an interconnection network, which contributes with a substantial share to chip area, energy consumption, and system performance. The use of the interconnect is determined by the memory subsystem. By studying the communication patterns of the coherence protocol, we have come up with a smart network design that reduces both energy and area in the interconnect, and improves system performance.

Our work was inspired by the observation that most of the traffic follows a request-reply pattern, which helps anticipate the path most replies will follow. We have used that information to propose a mechanism called Reactive Circuits based on reserving network resources and dynamically building the circuit for the reply while the request travels through the network. Consequently, reply messages with a set-up circuit can go through the router in a single cycle, compared with the four cycles needed in the baseline router. Guaranteeing complete circuits for data messages has also enabled us to predict when they will reach their destination, and elegantly eliminate the need for their acknowledgement. To evaluate the proposal, we have performed full-system simulation with realistic parallel and multiprogrammed workloads. For a 64-core chip, where the NoC has more impact, our proposal with complete reactive circuits achieves an average energy reduction of 20.8% at the NoC, routers have 5.8% smaller area, and system performance improves by 4.8%.

7. Acknowledgements

This work was supported in part by grants TIN2013-46957-C2-1-P, Consolider NoE TIN2014-52608-REDC (Spanish Gov.), and gaZ: T48 research group (Aragn Gov. and European ESF), and FPU12/02553.

8. Bibliography

[1] D. Sanchez, G. Michelogiannakis, C. Kozyrakis, An analysis of on-chip interconnection networks for large-scale chip multiprocessors, ACM Transactions on Architecture and Code Optimization 7 (1) (2010) 4:1–4:28. doi:10.1145/1756065.1736069.
URL http://doi.acm.org/10.1145/1756065.1736069

[2] M. Ortin, D. Suarez, M. Villarroya, C. Izu, V. Vinals, Dynamic construction of circuits for reactive traffic in homogeneous CMPs, in: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, 2014, pp. 1–4. doi:10.7873/DATE.2014.254.

[3] R. Kumar, V. Zyuban, D. M. Tullsen, Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling, in: Proceedings of the 32nd annual international symposium on Computer Architecture, ISCA '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 408–419. doi:http://dx.doi.org/10.1109/ISCA.2005.34.
URL http://dx.doi.org/10.1109/ISCA.2005.34

[4] F. Palumbo, D. Pani, A. Congiu, L. Raffo, Concurrent hybrid switching for massively parallel systems-on-chip: the cyber architecture, in: Proceedings of the 9th conference on Computing Frontiers, CF '12, ACM, New York, NY, USA, 2012, pp. 173–182. doi:10.1145/2212908.2212933.
URL http://doi.acm.org/10.1145/2212908.2212933

[5] J. Duato, P. Lopez, F. Silla, S. Yalamanchili, A high performance router architecture for interconnection networks, in: Proceedings of the International Conference on Parallel Processing, Vol. 1, 1996, pp. 61–68 vol.1. doi:10.1109/ICPP.1996.537144.
URL http://dx.doi.org/10.1109/ICPP.1996.537144

[6] A. Abousamra, A. K. Jones, R. Melhem, Proactive circuit allocation in multiplane NoCs, in: Proceedings of the 50th Annual Design Automation Conference, DAC '13, ACM, New York, NY, USA, 2013, pp.

35:1–35:10. doi:10.1145/2463209.2488778.
URL http://doi.acm.org/10.1145/2463209.2488778

[7] A. Abousamra, R. Melhem, A. Jones, Deja-vu switching for multiplane NoCs, in: Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS), 2012, pp. 11 –18. doi:10.1109/NOCS.2012.9.

[8] N. D. E. Jerger, L.-S. Peh, M. H. Lipasti, Circuit-switched coherence, in: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip, NOCS '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 193–202.
URL http://dl.acm.org/citation.cfm?id=1397757.1397999

[9] A. Abousamra, A. Jones, R. Melhem, Codesign of NoC and cache organization for reducing access latency in chip multiprocessors, IEEE Transactions on Parallel and Distributed Systems 23 (6) (2012) 1038–1046. doi:10.1109/TPDS.2011.238.

[10] D. Kline, Jr., K. Wang, R. Melhem, A. K. Jones, Mscs: Multi-hop segmented circuit switching, in: Proceedings of the 25th Edition on Great Lakes Symposium on VLSI, GLSVLSI '15, ACM, New York, NY, USA, 2015, pp. 179–184. doi:10.1145/2742060.2742087.
URL http://doi.acm.org/10.1145/2742060.2742087

[11] A. Mazloumi, M. Modarressi, A hybrid packet/circuit-switched router to accelerate memory access in NoC-based chip multiprocessors, in: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE, Grenoble, France, March 9-13, 2015, pp. 908–911.
URL http://dl.acm.org/citation.cfm?id=2757023

[12] S. Liu, A. Jantsch, Z. Lu, Parallel probe based dynamic connection setup in TDM NoCs, in: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1–6. doi:10.7873/DATE.2014.252.

[13] L.-S. Peh, W. Dally, Flit-reservation flow control, in: High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on, 2000, pp. 73–84. doi:10.1109/HPCA.2000.824340.

[14] P. Gaughan, S. Yalamanchili, A family of fault-tolerant routing protocols for direct multiprocessor networks, IEEE Transactions on Parallel and Distributed Systems 6 (1995) 482–497. doi:10.1109/71.382317.

[15] C.-Y. Lee, N. Jha, Variable-pipeline-stage router, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 21 (9) (2013) 1669–1682. doi:10.1109/TVLSI.2012.2217401.

[16] R. Mullins, A. West, S. Moore, The design and implementation of a low-latency on-chip network, in: Proceedings of the 2006 Asia and South Pacific Design Automation Conference, ASP-DAC '06, IEEE Press, Piscataway, NJ, USA, 2006, pp. 164–169. doi:10.1145/1118299.1118348.
URL http://dx.doi.org/10.1145/1118299.1118348

[17] R. Mullins, A. West, S. Moore, Low-latency virtual-channel routers for on-chip networks, in: Proceedings of the 31st annual international symposium on Computer architecture, ISCA '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 188–.
URL http://dl.acm.org/citation.cfm?id=998680.1006717

[18] L.-S. Peh, W. J. Dally, A delay model and speculative architecture for pipelined routers, in: Proceedings of the 7th International Symposium on High-Performance Computer Architecture, HPCA '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 255–.
URL http://dl.acm.org/citation.cfm?id=580550.876446

[19] A. Kumar, P. Kundu, A. Singhx, L.-S. Peh, N. Jha, A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS, in: 25th International Conference on Computer Design, ICCD, 2007, pp. 63–70. doi:10.1109/ICCD.2007.4601881.

[20] J. Kim, Low-cost router microarchitecture for on-chip networks, in: 42nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-42, 2009, pp. 255–266.

[21] J. Mische, T. Ungerer, Low power flitwise routing in an unidirectional torus with minimal buffering, in: Proceedings of the Fifth International Workshop on Network on Chip Architectures, NoCArc '12, ACM, New York, NY, USA, 2012, pp. 63–68. doi:10.1145/2401716.2401730.
URL http://doi.acm.org/10.1145/2401716.2401730

[22] M. Zhang, K. Asanovic, Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors, in: Proceedings of the 32nd annual international symposium on Computer Architecture, ISCA

'05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 336–345. doi:10.1109/ISCA.2005.53.
URL http://dx.doi.org/10.1109/ISCA.2005.53

[23] C. Seiculescu, S. Volos, N. Khosro Pour, B. Falsafi, G. De Micheli, CCNoC: On-Chip Interconnects for Cache-Coherent Manycore Server Chips, in: Proceedings of the Workshop on Energy-Efficient Design (WEED 2011), 2011.

[24] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese, Piranha: a scalable architecture based on single-chip multiprocessing, in: Proceedings of the 27th annual international symposium on Computer architecture, ISCA '00, ACM, New York, NY, USA, 2000, pp. 282–293. doi:10.1145/339647.339696.
URL http://doi.acm.org/10.1145/339647.339696

[25] Tilera, TILEPro64http://www.tilera.com/products/processors/TILEPro_Family (Last access November 2015).
URL http://www.tilera.com/products/processors/TILEPro_Family

[26] Intel, Intel Xeon Phihttp://www.intel.es/content/dam/www/public/us/en/documents/datasheets/xeon-phi-coprocessor-datasheet.pdf (Last access November 2015).
URL http://www.intel.es/content/dam/www/public/us/en/documents/datasheets/xeon-phi-coprocessor-datasheet.pdf

[27] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling, IEEE Journal of Solid-State Circuits 46 (1) (2011) 173 –183. doi:10.1109/JSSC.2010.2079450.

[28] A. Raghavan, C. Blundell, M. M. K. Martin, Token tenure and PATCH: A predictive/adaptive token-counting hybrid, ACM Transactions on Architecture and Code Optimization 7 (2) (2010) 6:1–6:31.

doi:10.1145/1839667.1839668.
URL http://doi.acm.org/10.1145/1839667.1839668

[29] R. Fernandez-Pascual, J. Garcia, M. Acacio, J. Duato, A fault-tolerant directory-based cache coherence protocol for CMP architectures, in: IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008, pp. 267–276. doi:10.1109/DSN.2008.4630095.

[30] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, B. Hughes, Cache hierarchy and memory subsystem of the AMD opteron processor, Micro, IEEE 30 (2) (2010) 16–29. doi:10.1109/MM.2010.31.

[31] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: A full system simulation platform, Computer 35 (2) (2002) 50 –58. doi:10.1109/2.982916.

[32] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, D. A. Wood, Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, SIGARCH Computer Architecture News 33 (2005) 92–99. doi:http://doi.acm.org/10.1145/1105734.1105747.
URL http://doi.acm.org/10.1145/1105734.1105747

[33] N. Agarwal, T. Krishna, L.-S. Peh, N. Jha, GARNET: A detailed on-chip network model inside a full-system simulator, in: IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2009, pp. 33 –42. doi:10.1109/ISPASS.2009.4919636.

[34] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, V. Stojanovic, DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling, in: Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip, NOCS '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 201–210. doi:10.1109/NOCS.2012.31.
URL http://dx.doi.org/10.1109/NOCS.2012.31

[35] C. Bienia, S. Kumar, J. P. Singh, K. Li, The PARSEC benchmark suite: characterization and architectural implications, in: Proceedings of the

17th international conference on Parallel architectures and compilation techniques, PACT '08, ACM, New York, NY, USA, 2008, pp. 72–81. doi:http://doi.acm.org/10.1145/1454115.1454128.
URL http://doi.acm.org/10.1145/1454115.1454128

[36] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, The SPLASH-2 programs: characterization and methodological considerations, in: Proceedings of the 22nd annual international symposium on Computer architecture, ISCA '95, ACM, New York, NY, USA, 1995, pp. 24–36. doi:http://doi.acm.org/10.1145/223982.223990.
URL http://doi.acm.org/10.1145/223982.223990

[37] Standard Performance Evaluation Corporation (SPEC), SPEC CPU2006, http://www.spec.org/cpu2006/ (Last access November 2015).
URL http://www.spec.org/cpu2006/

[38] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, 1991.
URL https://books.google.es/books?id=HetQAAAAMAAJ

[39] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, A. Agarwal, On-chip interconnection architecture of the tile processor, IEEE Micro 27 (5) (2007) 15–31. doi:10.1109/MM.2007.89.
URL http://dx.doi.org/10.1109/MM.2007.89