Dynamic Construction of Circuits for Reactive Traffic in Homogeneous CMPs

Marta Ortín[1], Darío Suárez[1], María Villarroya[1], Cruz Izu[2], Víctor Viñals[1]
[1]Departamento de Informática e Ingeniería de Sistemas, i3A, University of Zaragoza, Spain.
Email: {ortin, dario, mvg, victor}@unizar.es
[2]School of Computer Science, University of Adelaide, Australia. Email: cruz@cs.adelaide.edu.au

***Abstract*—Networks on Chip (NoCs) have a large impact on system performance, area and energy. Considering the characteristics of the memory subsystem while designing the NoC helps identify improvement opportunities and build more efficient designs. Leveraging the frequent request-reply pattern, our proposal dynamically builds the reply path in advance, is able to share circuits between messages, and even removes some implicit replies, significantly reducing NoC latency. A careful implementation of this circuit reservation mechanism achieves an average 17% reduction in router energy consumption, 8% smaller router area and a 2% system performance increase, compared with its baseline counterpart.**

I. Introduction

The design of multicore Networks on Chip (NoCs) can take advantage of the reactive nature of the traffic among nodes. This paper presents and evaluates a novel approach of dynamic virtual circuits for homogeneous chip multiprocessors (CMPs) with 16 and 64 cores connected by a mesh.

Analysing the coherency protocol in a standard wormhole 4-stage pipeline router, we detected that the request-reply pattern dominates over the rest. In average, 53% 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 channels, removing those stages from the critical path. We also observed that the network is lightly loaded (nodes inject, in average, less than 4 flits every 100 cycles) suggesting it will be feasible to reserve resources for longer periods of time.

This simple yet efficient dynamic circuit approach reduces network latency and achieves better performance than the baseline router. At the same time, it significantly reduces router area and energy consumption by removing buffer space. These results emphasize the importance of considering the system as a whole and studying how all the elements interact [1].

II. State of the art

Several works have proposed hybrid packet-circuit switching techniques to speed up certain messages. Most mechanisms establish circuits between nodes using dedicated networks. Some proposals have separate networks for packet and circuit switched messages [2], [3], while others implement a single network that supports both types of traffic [4], [5]. A different technique preallocates resources in advance to allow faster data transmission [6], [7].

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 [8], [9]. These routers are more complex and may require reduced network frequency or result in energy and performance penalizations when the implemented shortcuts cannot be used.

Contrary to previous approaches, our work does not require extra networks, additional messages, gathering statistics, or modifying the coherence protocol. Our proposal leverages the memory hierarchy behaviour to efficiently reserve network resources in advance with minimal changes in the routers.

978-3-9815370-2-4/DATE14/©2014 EDAA

TABLE I: 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
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 II: Messages generated by the coherence protocol.

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

III. Dynamic circuit construction

This section presents the characteristics of the CMP and the baseline interconnection network. After that, it explains the mechanism to dynamically build and use circuits to reduce communication latency.

A. 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. Table I summarizes the key parameters of the architecture and Table II details the messages exchanged by our MESI coherence protocol.

The baseline NoC is built as a mesh with simple 4-stage routers, dimension order routing and wormhole flow control. Table III includes the detailed configuration of the baseline NoC.

TABLE III: 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 1-phase VC/switch allocators, Round-robin 5-flit buffers per VC, enough to store a whole message
Links	16B flits, 1-cycle latency

B. Building circuits ahead of time

When a request reaches its destination, we already know that a reply is going to be sent back to the requestor. If we send the head of the message in advance, it can reserve the resources along the way in parallel with the L2 access. The drawback of this method is that the L2 access is too fast compared to the time it would take to set up the circuit (7 cycle L2 hit access versus an average of 10 and 24 cycles to traverse the network in 16 and 64-core chips, respectively).

To overcome this problem, the request, which is composed of one single flit, reserves the reply circuit as it is travelling towards the destination. Using dimension order routing (DOR), request and reply follow XY paths, which are disjoint (unless travelling in one dimension only). To apply our method, requests use XY routing and replies use YX routing.

Requests go through the 4 original stages of the router (see Table III). In parallel with VC allocation, the circuit is built for the reply. During that process, the necessary information to identify the circuit is stored in the router (requestor identifier and 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.

Out of the message types in Table II, the method creates circuits for data sent from the L2 to the L1 after a request (L2_Replies) and replacement acknowledgements (L1_WB_ACK), which are 52% of all reply messages. Invalidation acknowledgements (L1_INV_ACK), main memory replies (MEMORY), and direct data transfers between L1 caches (L1_TO_L1) are only 5.7% of the reply messages. 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, but they do not follow the same path as the request and reply between L1 and L2. Therefore, it is not possible to use a previous message to build a circuit for them.

C. 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 alternatives:

- Allow fragmented circuits keeping the partial path reserved, and attempt to reserve the rest of the path after the next hop.
- Allow only complete circuits, so that any lack of resources will 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 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: resources are not enough to exploit the full potential of the proposal. Therefore, in this alternative we include an additional VC to increase the number of simultaneous circuits, ending up with a total of 3 VCs in the reply virtual network.

Building only complete circuits allows to implement many simplifications in the router. We guarantee that a message with a circuit built has all the resources it needs from source to destination. Hence, it will never get blocked in the network. This has two beneficial consequences: 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 because flits will just go through the router without stopping. We experimentally explored the best number of simultaneous circuits built per input port and set it to 5. Figure 1 presents the modified router that implements the reservation of complete circuits.

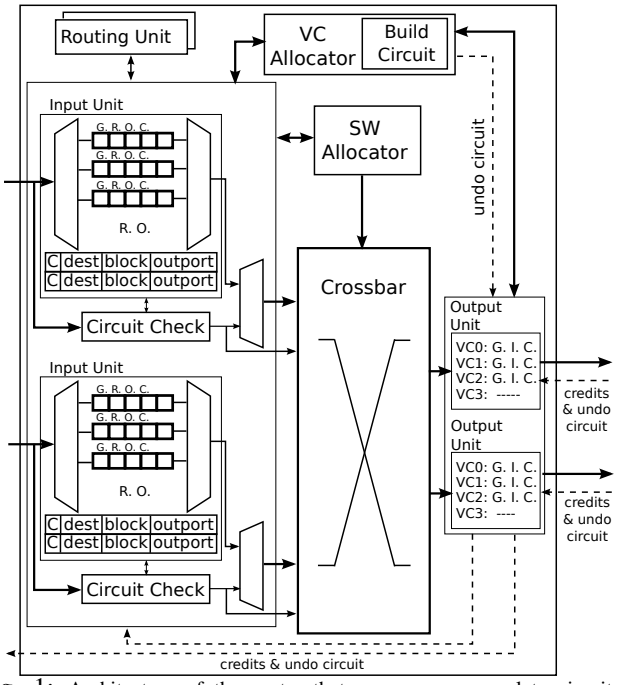


Fig. 1: Architecture of the router that can reserve complete circuits. It includes "Circuit Check" logic at the input units and a "Build Circuit" module in the VC allocator. In this drawing, two simultaneous circuits can be built per port. VCs at the input units store global stage (G), route (R), output VC (O) and credit count (C). Circuit information includes circuit-built bit (C), destination identificator (dest), cache line address (block) and output port (outport). Credits may carry undo-circuit information. At the output units, they store global state (G), input VC (I) and credit count (C).

D. 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. With fragmented circuits, a message that had a circuit might arrive at a router where there is no built circuit. When that happens, it will just be stored in the VC and go through the usual 4 stages of the router.

Even if 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.

E. Undoing circuits before they are used

We must undo a circuit before it gets used under several situations. The coherence protocol lets an L2 cache 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 will never be used. When a complete circuit cannot be fully booked, reservations in previous routers must also be undone.

In 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.

F. Reusing 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 method and try to find other messages that can reuse the circuits. When a reply that does not have a circuit built is about to leave the network interface, it checks if there is any circuit it could use to get closer to its destination. In that case, the message becomes a *scrounger* message that uses the circuit until an intermediate destination. At that point, the network interface will re-inject the message so 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.

G. Eliminating coherence messages

Studying the coherence protocol while designing the NoC has allowed us to notice a rewarding effect of our mechanism. We already mentioned that we cannot build a circuit for the L1_DATA_ACK reply messages that are sent from the L1 to the L2 after the L2_Reply (see Section III-B). However, if this L2_Reply uses a circuit to get to the L1 requestor, we can predict exactly how long it will take the data to reach its destination and inform 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.

IV. Evaluation

This section presents the simulation methodology and the main results of our proposal.

A. Simulation framework and workloads

We use Simics, GEMS and an extended version of Garnet to carefully model all the components of the chip and perform full system simulation. To get the timing, area and energy expended by the network we use DSENT, a state-of-the-art circuit modelling tool. We use 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 and SPLASH2 with scaled inputs from PARSEC 3.0. 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 and bind each application to a different core. To build the workloads for the 16-core chip, we randomly distribute the applications to build 20 different mixes. For the 64-core chip, we use each application 4 times, and again build 20 different mixes. To perform the evaluation, we warm up the caches for 200 million cycles and simulate for 500 million cycles.

B. Results

A smart architecture design must optimize performance while considering power and area. Therefore, we present a comprehensive evaluation of our proposal that includes all three aspects.

Figure 2 shows the energy expended at the router when using our mechanism with respect to the baseline router with XY routing for requests and YX routing for replies (for the multiprogrammed workloads, we measure energy per instruction). We include results for 16 and 64-core configurations, with fragmented circuits, complete circuits (5 per input port), circuit reuse and eliminating coherence messages in the last two cases. As we anticipated, increasing reply virtual channels from 2 to 3 in the fragmented case results in higher energy consumption. However, the reduction of buffer storage in the complete case achieves significant energy reductions, especially with 64 nodes (an average of 18% with parallel applications).

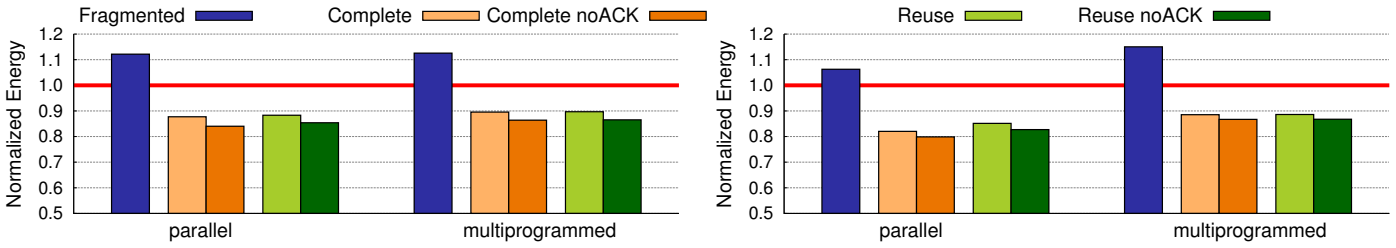
Reusing circuits does not always provide better results because scrounger messages can sometimes delay the message the circuit was originally built for. As a consequence, some circuits are reserved for longer periods of time, keeping resources busy. Removing coherence messages always permits further energy savings due to the reduced amount of network traffic.

Regarding the area needed to implement the proposal, the size of the router increases by 23% with fragmented circuits. On the other hand, removing buffer space at the virtual channels when using complete circuits allows us to reduce the router area by 8%. Complexity at the network interfaces increases slightly due to logic to build and use the circuits and store their identification data.

Figure 3 shows the performance speedup achieved with our mechanism measured in execution time for parallel applications and number or instructions for the multiprogrammed workloads. We get higher speedups with fragmented circuits because more messages reduce their latency, although this improvement is costly in terms of energy. On the other hand, complete circuits exhibit minimal speedup but achieve significant energy savings. Coupling the reservation of complete circuits with the elimination of some coherence messages results in the largest improvements.

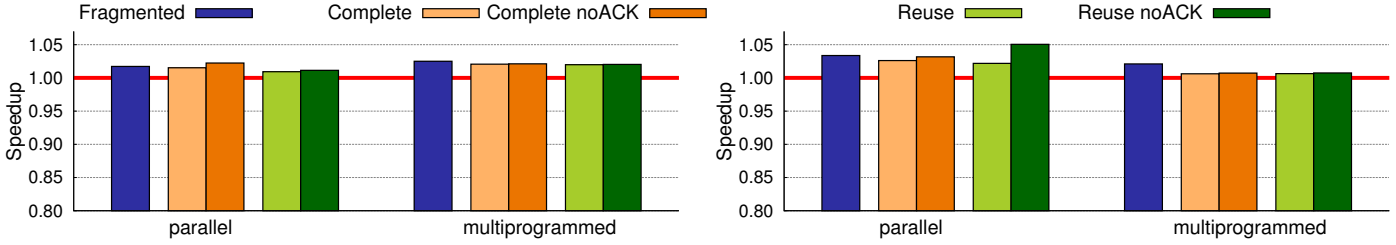
Our proposal has slimmer benefits for multiprogrammed workloads, especially in the 64-core configuration. That is because these workloads access main memory more often, so the tiles with memory controllers need to handle more traffic. Since we do not build circuits for the MEMORY replies, all that traffic lies on a single virtual channel, creating some congestion.

Performance improvements are a direct consequence of the network latency reduction achieved by our mechanism. This latency is a combination of the average time messages take to reach their destination and the queuing latency, that is, time spent in the network interface waiting to access the network. Our mechanism always improves the former value, achieving a peak latency reduction of 26% with multiprogrammed workloads executed on 16 cores and an overall average of 16%. This improvement comes from the replies that quickly travel through a complete circuit: 68% with 16 cores and 36% with 64 cores. The percentage is considerably lower in the second case because it is more complicated to find idle resources along longer paths. The queuing latency suffers with



(a) 16 cores (b) 64 cores

Fig. 2: Average network energy expended with circuit reservations measured as total energy for parallel applications and energy per instruction for multiprogrammed workloads, with respect to the baseline results (baseline router with XY routing for requests and YX routing for replies). The lower the bars, the better.



(a) 16 cores (b) 64 cores

Fig. 3: Average speedup achieved with circuit reservations measured as number of execution cycles for parallel applications and number of completed instructions for multiprogrammed workloads, normalized to the baseline results (baseline router with XY routing for requests and YX routing for replies).

the implementation of our mechanism because all replies that do not have a circuit must contend for a single virtual channel, causing our overall average latency reduction to drop to 7%. Techniques that reduce traffic in that virtual channel, such as eliminating redundant coherence messages, help reduce this undesirable effect and improve performance.

Comparing the results for the 16-node and the 64-node NoCs, we see that both speedup and energy savings are higher in the latter case, even though less percentage of replies manage to get a complete circuit. That is because the impact of the NoC is bigger on a larger chip: more messages are sent per cycle and their latency is larger. This shows that the method scales well and shows a lot of promise for future larger chips.

We also checked that the effectiveness of our mechanism is independent from the L2 latency, even though circuits remain built during the L2 cache access. The difference in speedups for latencies of 1, 4, 7 (reported by DSENT), 12, and 20 cycles in 16 and 64-core chips with the best configuration (building complete circuits and removing coherence messages) is always smaller than 0.01.

V. Conclusions

This paper was inspired by the observation that most of the traffic follows a request-reply pattern, which helps anticipate the path for most replies. We have used this information to reserve network resources and dynamically build 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 4 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, our proposal achieves an average energy reduction of 17% at the router, 8% smaller area, and speedups of 2%. Results are better for the 64-core configuration than for the 16-core one and are not dependent on L2 access latency, which shows that the mechanism scales well and will benefit future designs.

Acknowledgments

This work was supported in part by grants TIN2010-21291-C02-01 (Spanish Government, European ERDF), gaZ: T48 research group (Aragón Government and European ESF), Consolider CSD 2007-00050 (Spanish Government), and HiPEAC-3 NoE (European FP7/ICT 217068).

References

[1] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in Multi-Core architectures: Understanding mechanisms, overheads and scaling," in *Int. Symp. on Computer Architecture*, 2005, pp. 408–419.

[2] F. Palumbo, D. Pani, A. Congiu, and L. Raffo, "Concurrent hybrid switching for massively parallel systems-on-chip: the cyber architecture," in *Procs of the 9th conf. on Computing Frontiers*, 2012, pp. 173–182.

[3] J. Duato, P. Lopez, F. Silla, and S. Yalamanchili, "A high performance router architecture for interconnection networks," in *Procs of the Int. Conf. on Parallel Processing*, 1996, pp. 61–68 vol.1.

[4] N. D. E. Jerger, L.-S. Peh, and M. H. Lipasti, "Circuit-switched coherence," in *Procs of the Int. Symp. on Networks-on-Chip*, 2008, pp. 193–202.

[5] A. Abousamra, A. Jones, and R. Melhem, "Codesign of NoC and cache organization for reducing access latency in chip multiprocessors," *IEEE Trans. on Parallel and Distributed Systems*, pp. 1038–1046, 2012.

[6] L.-S. Peh and W. Dally, "Flit-reservation flow control," in *Int. Symp. on High-Performance Computer Architecture*, 2000, pp. 73–84.

[7] C. Lee and N. Jha, "Variable-pipeline-stage router," in *IEEE Trans. on Very Large Scale Integration Systems*, 2012, pp. 1–1.

[8] R. Mullins, A. West, and S. Moore, "The design and implementation of a low-latency on-chip network," in *Procs. of the Asia and South Pacific Design Automation Conference*, 2006, pp. 164–169.

[9] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Procs of the Int. Symp. on High-Performance Computer Architecture*, 2001, pp. 255–.