On-Chip Ring Network Designs for Hard-Real Time Systems

Miloš Panić[§,‡], Germán Rogríguez[⋆], Eduardo Quiñones[§], Jaume Abella[§], Francisco J. Cazorla[§,*]
[§]Barcelona Supercomputing Center (BSC-CNS) [‡]Universitat Politecnica de Catalunya (UPC)
[⋆]IBM Research, Zurich [*]Spanish National Research Council (IIIA-CSIC), Spain
{milos.panic,eduardo.quinones,jaume.abella,francisco.cazorla}@bsc.es {zrlrod}@ch.ibm.com

ABSTRACT

Rings have been extensively used in high-performance systems to improve performance and scalability, and to reduce cost, energy and design effort. However, in the real-time domain, they have not been thoroughly analyzed to provide worst-case time bounds. We propose several on-chip ring designs in shared-memory multicore processors that enable the computation of trustworthy upper bounds to the time required for a packet to traverse the ring, which is a fundamental requirement to enable their use in real-time systems.

1. INTRODUCTION

Critical Real-Time Embedded System (CRTES) industry requires increasing computing power to provide more and more functionalities, to keep the competitive edge. For instance, car manufacturers already incorporate systems like electronic parking brakes and x-by-wire technology which have high guaranteed performance requirements. Similarly, a modern aircraft requires millions of lines of code just for on-board control functions such as guidance, navigation and anti-collision systems among others. Those high performance requirements can be met by designing more complex processors, e.g. with out-of-order execution and higher clock frequency. However, applying these solutions to CRTES design is not feasible, because (1) they could introduce timing anomalies [1] due to their non-deterministic run-time behavior and (2) they have high energy requirements that affect the low-power constraints and severe cost limitations of common embedded systems.

Multicores have found their use as means to provide the required high performance. Multicores offer better performance per watt than single-core processors while maintaining a simple processor design. Moreover, multicores enable co-hosting several applications, which allows scheduling a high number of tasks on a single processor so that the hardware utilization is maximized while cost, size, weight and power requirements are reduced.

However, CRTES require guarantees on the timing correctness of the system and multicores timing is harder to analyze than that for single-cores. The source of this complexity are *inter-task interferences* when accessing hardware shared resources such as the interconnection network, shared caches, main memory, etc. Inter-task interferences appear when several tasks, sharing a resource, try to access it at the same time. To handle contention, an arbitration mechanism is required, potentially affecting the execution time of running tasks. As a result, providing a meaningful timing analysis becomes extremely difficult because the execution time, and the *Worst-Case Execution Time* (WCET) estimate, of a task may change depending on the other tasks running simultaneously. Hence, in CRTES, one of the main goals in the design of any Interconnection Network is to enable the computation of the *Worst-Case Traversal Time* (WCTT), that is, the time required for a request to traverse the network, including both inter-task interference delay and the intrinsic delay (the one actually required to traverse each node/link in the path from a source to a target node).

Rings have been used in recent commercial designs in mission-critical systems [2], and are widely used in existing high-performance multicore processors, such as in the IBM POWER4[3] and POWER5[4], in which a ring network is used for intra-core coherence, in the IBM/Sony/Toshiba Cell [5], and are being considered by Intel for next-generation multicores consisting of 8 to 16 cores[6]. Larger ring-based systems can be built by using a hierarchy, such as clustering or even a ring-of-rings. In that direction, several studies show that multi-ring configurations scale to large systems and provide a good area-performance trade-off while retaining many of the advantageous features of simpler rings arrangements [7][8][9].

While ring-networks in high-performance systems mainly aim at providing high average performance, their use in CRTES imposes different requirements. First, rings, as any other interconnection network, suffer from inter-task interferences that can significantly increase WCET estimates of tasks or simply make the execution time of tasks unbounded. CRTES require WCET estimates to be as low as possible. Rings are required to reduce the WCTT of requests sent through the network as a means to reduce WCET estimates obtained for tasks running on a multicore connected with rings. Second, *Time composability* is a property by which the WCET estimates computed for a task in a multicore must hold for any workload in which the task is executed. This requires that the WCTT obtained for the packets of a task must be independent of the load that other tasks put in the ring. And third, hardware complexity and power are severely constrained in CRTES, so rings design must be kept simple.

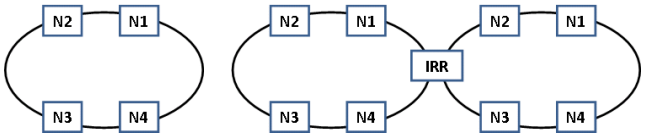
It is important to remark that providing a guaranteed bandwidth allocation for each program using the ring is not enough for WCET analysis techniques. The reason is that QoS techniques providing such guarantees do not provide a WCTT *for every request* a program sends to the ring as needed for WCET analysis.

This paper proposes several ring designs in shared-memory multicore processors adhering to the aforementioned constraints.

1) We initially propose a controlled injection-rate ring design in which, by controlling the rate at which packets are injected in the ring, inter-task interferences, and hence the WCTT, can be bound. This significantly reduces the buffering and control-flow logic overhead.

2) With the same aim, we apply TDMA (Time Division Multiple Access), a technique that splits the access to links

RTNS 2013 October 16 - 18 2013, Sophia Antipolis, France



(a) single ring (b) multi-ring network

Figure 1: Ring setups

into slots assigned to every node, such that the TDMA slots that receive a packet from a source to a destination are synchronized such that a packet in a node N_i arrives to node N_{i+1} by the time its TDMA window starts in N_{i+1}. Unlike the previous design, this requires global synchronization.

In all cases, we derive the WCTT analytically and compute WCET estimates using a commercial WCET analysis tool, RapiTime [10]. We focus on processor setups in which rings are used to connect a variable number of cores, from 4 to 16, to a shared on-chip memory. We consider two network topologies: single rings and multi-rings.

2. BACKGROUND CONCEPTS

2.1 Time Composability

The timing analysis of different functions of a CRTES is typically performed by different vendors, in parallel, without knowledge of the other functions that will run in the system. This issue is particularly critical in multicores where multiple functions are run concurrently. In order to allow trustworthy and independently performed timing analysis, *time composability* is needed. Time composability holds when the timing behavior of a function (in terms of its known time lower and upper bounds) is not affected by the execution of others. Time composability reduces the cost of system integration and qualification, which is one of the most critical challenges faced by system developers.

Several hardware techniques have been developed in order to enable time composability, and thus the use of existing timing analysis methods in multi-cores [11][12][13]. The basic approach to reach that goal is to provide an upper bound to the effect that interactions in access to resources cause in the WCET estimate of the program. In the case of the interconnection networks, rings in our case, every time the program has to use the ring for any communication, it is required an upper bound for the WCTT so that such bound can be used during timing analysis making WCET estimates composable. That is, the WCTT computed for a task should be independent of the ring load introduced by other tasks running concurrently. Note that it is not enough to provide an average guaranteed bandwidth allocation for the program, but to provide an upper bound to the latency of every single request to the ring. Rings and particularly bi-directional rings have been thoroughly analyzed in terms of throughput and fairness [14, 15]. QoS techniques, even if very varied and suited to different traffic scenarios [16, 17], do not attempt to provide an upper bound for the delays, but a best effort tradeoff, trying to reconcile (i) obtaining the maximum throughput possible while (ii) still achieving fairness and providing the capability of prioritized traffic. In the CRTE domain, simplicity, cost and analyzability are not only desirable features, but requirements that differentiate our work form previous ones.

2.2 Ring-based Networks

In our ring network, nodes comprise the router that communicates the node to the ring and a processing or memory element (called PME), see Figure 2(a). The latter could be either a processor core, main memory, I/O, etc. Each router may contain buffers to store flits temporarily, and some logic to determine which of the buffers is granted access to transmit through the output port to the next node. However, in the rest of this work we assume simple routers where each cycle up to one flit arrives to a router, which will forward such flit during the next cycle either to the following node or to its PME. This way only a one-entry buffer is required per router. Note that if a PME fails to inject a flit, it will keep it until the router does not receive incoming traffic and can, therefore, accept and forward such flit.

Unidirectional rings have simple interfaces that allow them to be clocked at fast rates. Each router has only two input ports (one for the traffic injected by the PME and one for the (transient) traffic coming from the previous node) and one output port, allowing rings to have wide data paths while still consuming low power.

Application messages are split into smaller units called packets (also called segments or frames depending on the technology). Packets might need to be further split into fixed-length elements called flits. Flits are *flow-controlled* units, i.e., a flit is the minimum transfer unit that can be either completely transmitted or not at all at the other end, but cannot be interleaved with other data transfers. Buffer sizes at the receiving side of the link must be sized to accommodate at least one flit. Depending on the flit length and link width, interconnection networks may transmit a full flit at once or may need to further split it into fixed-length elements called *phits*. The length of a *phit* must be such that it does not exceed link width. There is an obvious trade-off between the width of the links and the number of phits required per flit.

In general-purpose routers, an arbiter grants access to either an incoming (traversing) flow or to the flow coming from the PME attached to the node. If the traversing traffic is prioritized over the PME inserted traffic, there is no upper-bound delay for the PME inserted traffic. We refer to the insertion delay due to contention as D_{ins}^{cont}. This is so, because under heavy load conditions, a given flit may potentially wait an unlimited amount of time before it can be injected in the network. Conversely, if the PME-inserted traffic is prioritized over the traversing traffic, the problem is the opposite: there is not an upper-bound delay for the traversing traffic. We refer to the traversal contention delay as D_{trav}^{cont}. Hence, we have to arbitrate between incoming and traversing traffic in nodes to provide bounds to WCTT rather than simply prioritizing one over the other.

We consider two ring setups: (i) single-ring networks in which all nodes are connected through a single ring and (ii) multi-ring networks where independent single-ring networks are connected through a special node called inter-ring router (IRR), see Figure 1.

In the single-ring network, nodes are composed of the router that communicates the node to the ring and a PME, see Figure 2(a). Multi-ring networks comprise (regular) nodes and IRR nodes that switch flits across rings. IRR structure is shown in Figure 2(b). Differently to regular nodes, an IRR has two input ports (one from each of the two rings connected) and two output ports (also one to each of the two rings). Since flits from different input ports may arrive simultaneously and may target the same output port, some extra buffer capabilities with respect to those of the routers are required. In following sections, when presenting our designs, we determine the size of those buffers to allow a low WCTT that can be bound.

2.3 Performance Metrics

Network performance is generally summarized with two main metrics: average throughput and average delay under a certain load (packets generated per unit of time) and a particular communication pattern (usually a random distribution of destinations). These common metrics are unsuitable to estimate the WCTT in the CRTE domain as they do not provide us with upper bounds. Therefore, we will use the following metrics instead:

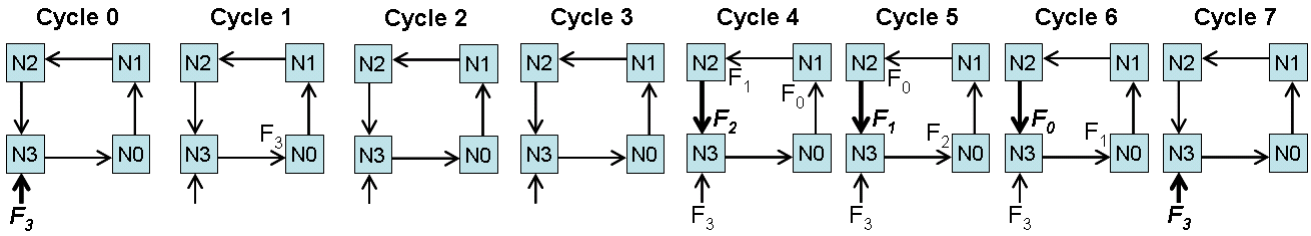
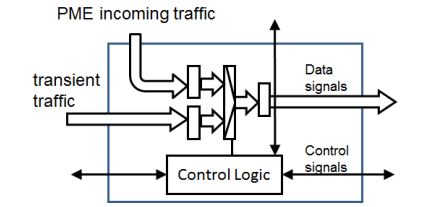
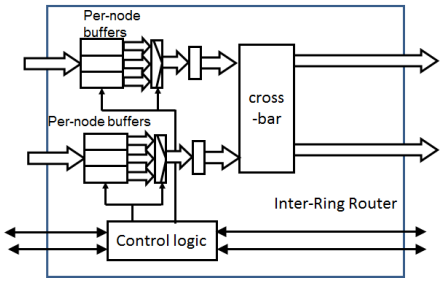


Figure 3: Example of a flit (F_3 in cycle 4) paying the maximum injection delay in a single-ring network with 4 nodes. Bold flits and arrows indicate which flit has been granted access through the N3 node.



(a) Router structure



(b) Inter-ring node structure

Figure 2: Router (top) and IRR structure (bottom)

- Maximum Theoretical Capacity (MTC): commonly computed as the summation of the throughputs of all links in the network.
- Maximum Workload Capacity (MWC): The average workload capacity for a given workload is the fraction of the MTC achieved when that workload runs on the network without WCET guarantees. The Maximum Workload Capacity is the maximum fraction of the MTC that any workload could achieve.
- Maximum Guaranteed Capacity (MGC): It is the fraction of the MTC on which WCET guarantees can be asserted. MGC is computed based on the WCTT of packets instead of its actual traversal time. The latter can only be computed at run time depending on the actual resource contention, and therefore cannot be used to derive a-priori WCET estimates.

3. SINGLE RINGS

In this paper, we use rings to connect cores to a shared on-chip memory. In our ring design, flits of the same packet can be sent from a node in non-consecutive cycles. This requires having routing information per flit. In an N-node ring, this translates into having $\lceil log_2(N_{sR}) \rceil$ bits identifying the target node, where N_{sR} is the number of nodes of the ring. In a 16-node ring with a data path of 128 bits, this is a 3% overhead in the number of wires. Note that the low number of ports and simple router design allow rings to have wide data paths. Further, the resequencing is not needed since flits arrive in order to the destination.

In our ring, in a given cycle, a router R_i can simultaneously send and receive flits. That is, let's assume that, in a given cycle n, there is a flit in the router (this could be a transient or injected flit). During cycle $n+1$ the stored flit can be sent to R_{i+1} while another transient flit from R_{i-1} or new flit from the PME is injected. This requires 3 buffers, see Figure 2(a).

We start the analysis focusing on a single-level N-node ring, which we call source Ring (sR). We present two alternative approaches for the single ring design with different effects on Worst-Case Traversal Time ($WCTT$). For the sake of clarity, and without loss of generality, in the description of the approaches, we assume that the time to traverse a router (D_{trav}^{router}) and the time to traverse a link (D_{trav}^{link}) is 1 cycle, meaning that we assume that each flit is composed of 1 phit. Generalizing the analytical models to support link and router latencies bigger than 1 (i.e. 1 flit is split into several phits and/or pipelined routers/links are used) is straightforward.

3.1 Controlled Injection-Rate Ring

This design aims at reducing the hardware cost in terms of buffering and switching capacity. To that end, traversing flits have priority over new injections, but each PME is limited to inject flits inside the network with a maximum rate. This guarantees existance of time slots for each PME to inject new flits in the ring, *by construction*.

MFII: Our approach consist in limiting each node not to inject a new flit from its PME before a *minimum flit injection interval* (MFII) has elapsed since the preceding flit was injected in the ring, with $MFII_{sR} = N_{sR}$, where N_{sR} is the number of nodes of the ring. Hence, the maximum flit injection rate is $\frac{1}{MFII_{sR}}$.

If all nodes in the ring inject flits without violating the $MFII_{sR}$ then, once a flit is ready to be sent in any node, in a time window of less than $2 \times MFII_{sR} - 1$ cycles, that node will find an empty time slot to send that flit. In fact, the flit has to wait $MFII_{sR}$ cycles to be injected in the ring since the node injected the previous flit. Enforcing this in all nodes guarantees that each node has a slot available every $MFII_{sR}$ cycles. Thus, the flit may have to wait up to $MFII_{sR} - 1$ extra cycles due to traversing flits from the remaining $N_{sR} - 1$ nodes. Routers use a multiplexer with priority for the traversing flits, so that if a flit is to be injected when a traversing flit arrives to the node from the ring, then the former is delayed.

This scenario is illustrated in the example in Figure 3. In this particular example, PME in node 3 (N3) injects a flit in cycle 0. Such flit travels to N0 the next cycle. Then it may be removed by N0 if its destination is the PME in N0, although this is irrelevant for the example. PME in N3 has another flit ready for sending, but cannot inject it until cycle 4 due to the required $MFII_{sR}$ cycles that must elapse between consecutive injections. Later, in cycle 3, PMEs in N0, N1 and N2 try to inject one flit each. Those flits are successfully injected so that they travel to the next node in the ring in cycle 4. Eventually, the PME in N3 tries to injecting its flit in cycle 4, but flits from other nodes arrive also to N3 in cycles 4, 5 and 6. Since flits in the ring have priority over flits to be injected, N3 does not accept the new flit from its PME until cycle 7, when no further flit from other nodes can arrive. Note that the soonest a flit from any other node (N2) could arrive is cycle 8, if N2 injects a new flit from its PME exactly $MFII_{sR}$ cycles after its previous injection (so N2 injections occur in cycles 3 and 7).

Deadlock avoidance: In standard rings deadlocks could occur if wait-for cycles appear and resources are not available to move flits towards their destination, where they will be consumed. Several ways exist to avoid deadlocks, such as the use of *virtual channels* [18], or by means of *bubble flow-control* [19, 20].

We use a bubble flow-control mechanism to avoid deadlocks. The idea behind it is to ensure that at any point in time, a free resource (a bubble) is present in the ring, guaranteeing that transient packets will be able to progress and be eventually consumed at their destinations. A conservative approach to keep this guarantee is to allow only an injection from a node *if and only if* the injection will leave a bubble. This is achieved in our case since every router has a buffer for the transient traffic and another for the packet to be injected. An injection of a new packet in a given cycle is allowed (a.k.a. the packet is sent through the link to the next node) only if no transient packet arrived by the end of the previous cycle. The prioritization of transient traffic ensures that once in the ring, transient packets will never be stopped, and the injection rate imposed to the nodes ensures that any single node will be able to inject within the next $2 \times MFII_{sR} - 1$ cycles.

WCTT. The WCTT includes (1) the traversal delay of the nodes and routers and (2) the inter-task interference delay. The principle of our Controlled Injection-Rate Ring is to make flits wait to get access to (be injected into) the ring such that once the flit starts traversing the ring it suffers no inter-task delays. Note that, in general, the WCET estimation needs to compute the WCTT twice for each communication: for the request and for the response, and the WCTT may differ for requests and responses based on their number of flits. For instance, in the case of a read operation, the request transaction contains a destination address while the response transaction contains the data read, potentially containing many more flits than the request.

Given that flits travel in order and that no contention can occur during the ring traversal, a given transaction can take as much as needed to inject all flits plus the time it takes the last flit to reach its destination. This can be expressed as follows: $WCTT_{CIR} = (N_{flts} \times WD_{injflt}) + WD_{trav}$, where N_{flts} stands for the number of flits in the transaction, WD_{injflt} is the worst delay between flit injections and WD_{trav} is the worst traversal time. Each of those parameters is obtained as follows:

- WD_{trav}: In our network, WD_{trav} can be simply computed as the number of nodes plus links between the source and destination nodes of a transaction. In the worst case, this number equals the sum the number of nodes and links in the network minus one. , while the response transaction contains the data read, potentially containing much more flits than the former. WD_{trav} is defined as follows $WD_{trav} = \left(D^{router}_{trav} + D^{link}_{trav}\right) \times H^s_d$ where D^{router}_{trav} and D^{link}_{trav} stand for the router and link traversal time respectively as explained before, and H^s_d is the number of hops between the source and destination nodes of the transaction, where $H^s_d \leq (N_{sR} - 1)$.
- WD_{injflt}: As explained before, some delay can occur between consecutive flit injections due to (i) the controlled injection rate and (ii) contention. The worst delay between consecutive flit injections is as follows: $WD_{injflt} = 2 \times MFII_{sR} - 1$.
- N_{flts}: the number of flits of a particular transaction depends on the number of bits to be sent, the header information required in each flit (e.g., whether it is the first, the last or an intermediate flit; destination node; etc.) and the width of the links. The number of flits is obtained as follows:

$$N_{flts} = \frac{Data_{bits}}{Link_{width} - Header_{bits}} \quad (1)$$

where $Data_{bits}$, $Link_{width}$ and $Header_{bits}$ stand for the number of bits to be transmitted, the width of the link in bits and the size of the header in bits respectively.

Performance metrics: MWC and MGC. The MGC is obtained as follows:

$$MGC = N_{sr} \times \frac{1}{2 \times N_{sr} - 1} \quad (2)$$

given that our design guarantees that all N_{sr} nodes can inject one flit each every $2 \times N_{sr} - 1$ cycles. So MGC decreases asymptotically down to 0.5. For 4- and 8-node rings MGC is 0.57 and 0.53 approximately.

The MWC of the ring is obtained when the requests of the tasks in each node are aligned such that, on every flit injection, after the node waits N_{sr} cycles, the ring is available for that task to inject a new flit. Note that, at run time a node injects a new flit as soon as there is free slot after waiting N_{sr} cycles. Hence, in the ideal scenario when nodes try to inject new flits *simultaneously*, injections occur every N_{sR} cycles so that MWC is as follows:

$$MWC = N_{sr} \times \frac{1}{N_{sr}} = 1 \quad (3)$$

Thus, *full utilization of the ring is feasible under heavy sustained loads.*

3.2 Rotating TDMA (*rtdma*) Ring

This design also aims at decreasing the WCTT at low cost. It builds upon TDMA and basically applies time sharing between the requests of the different contenders to the ring links. Time is divided into windows, in each of which, each contender is assigned a slot. When a request of a given contender becomes ready in that contender's slot, the request gets immediate access to its corresponding node. If the request becomes ready outside the contender's slot, it has to wait until the contender's upcoming slot.

In order to implement rTDMA in a ring-based network, in every link each node is given a window of the same duration (1 cycle in our case) every N_{sR} cycles in each link. TDMA windows are globally synchronized, such that once a flit from node N_i is inserted in the ring in cycle n, the TDMA window of node N_i in nodes N_{i+1}, N_{i+2}, ... is available in cycles $n+1$, $n+2$, ... In this way flits, once injected, reach its destination without any stall in the ring.

A router do not even need to control contention, since no flit can arrive to it in the cycle when the router aims to inject a new flit. However, global synchronization is needed to make *rtdma* work as described above. If this was not the case, then injection could occur in a node when another flit were traversing that particular node. Thus, the main difference between the controlled injection rate ring and the *rtdma* one is the fact that the former does not need any type of global synchronization (synchronization occurs locally at each node), whereas the latter needs global synchronization. Global synchronization has been shown to introduce complex implementation issues due to the complexity and power dissipation of strategies to mitigate clock skew[1] [21].

MFII: The longest time a flit may be delayed before being injected in the ring is $MFII_{sR} = N_{sR} - 1$ cycles. This would be the case if a flit became ready to be sent 1 cycle after the injection window. Hence, the maximum injection rate is $\frac{1}{MFII}$.

Deadlock Avoidance Control: No deadlock is possible under *rtdma*, as the cycle of injection books the whole path for the injected packet and guarantees its continuous transmission free of stalls at each hop until it's arrival at the destination.

[1]Clock skew stands for the time difference between the arrival of the clock signal to different processor components. Large clock skews impede processors to keep all their components operating synchronously.

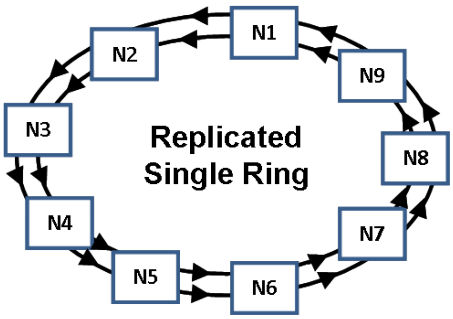


Figure 4: Replicated single-ring design

WCTT. WCTT is obtained as follows where N_{flts} and WD_{trav} are exactly the same as for the controlled injection rate ring.

$$WCTT_{rTDMA} = N_{flts} \times WD_{injflt} + WD_{trav} \quad (4)$$

However, $WD_{injflt} = N_{sR} - 1$ since there is no need for waiting N_{sR} extra cycles as for the controlled injection rate ring.

Performance metrics: MWC and MGC. With the *rtdma* ring, the MGC occurs when all PME are ready to send a flit in their time windows, so it is as follows:

$$MGC = N_{sr} \times \frac{1}{N_{sr}} = 1 \quad (5)$$

Since $MWC \geq MGC$ and $MGC = 1$, then $MWC = 1$ in the same scenario when $MGC = 1$.

3.3 Replicated Rings

So far we have considered single-ring setups. This has some implications on the *minimum flit injection interval* ($MFII_{sR}$), the worst delay between flit injections (WD_{injflt}) and the *worst traversal time* (WD_{trav}). For instance, in the case of a Controlled Injection-Rate Ring, these parameters are as follows:

$$MFII_{sR} = N_{sR}$$
$$WD_{injflt} = 2 \cdot MFII_{sR} - 1$$
$$WD_{trav} < \left(D_{trav}^{router} + D_{trav}^{link}\right) \times (N_{sR} - 1)$$

We may consider connecting the nodes with several rings (see Figure 4) such that we decrease $MFII_{sR}$, WD_{injflt}, and WD_{trav} of our single ring. For instance, we could set up two rings (R1 and R2) instead of only one and allow half of the nodes to use one ring (R1) and the other half to use the other ring (R2). The impact in hardware cost is obvious: we double the cost of the ring. However, both the $MFII_{sR}$ and WD_{injflt} would decrease. In particular, we would have the following:

$$MFII_{sR} = \left\lceil \frac{N_{sR}}{2} \right\rceil$$

and so WD_{injflt} would also halve due to its dependence on $MFII_{sR}$. WD_{trav} remains the same as for the single-ring setup.

Alternatively, we could set up two rings so that they forward flits in opposite directions (dual bidirectional ring) and PMEs use always the ring whose number of hops to the destination node is the shortest. This would decrease not only WD_{trav}, but also $MFII_{sR}$ and WD_{injflt} since any particular node in one of the rings will receive only flits from the closest half of the nodes since the other half will use the other ring. For instance, if we have such a dual bidirectional ring with 9 nodes, node 5 (N5) of ring 1 (R1) will only receive flits from PMEs in nodes N1, N2, N3 and N4, whereas N5 in ring 2 (R2) will only receive flits from PMEs in nodes N6, N7, N8 and N9. Under this ring configuration, we have:

$$MFII_{sR} = \left\lceil \frac{N_{sR}}{2} \right\rceil$$
$$WD_{trav} < \left(D_{trav}^{router} + D_{trav}^{link}\right) \times \left\lceil \frac{N_{sR}-1}{2} \right\rceil$$

with WD_{injflt} also halving due to its dependence on $MFII_{sR}$.

Overall, replicating rings is a viable alternative to reduce the different parameters of rings impacting WCET estimates. For the sake of simplicity and due to lack of space, we only focus on single rings in the rest of the paper.

4. MULTI-RING NETWORK

Single rings have a linear dependence on the number of nodes in the ring for both flit injection and network traversal. Therefore, multi-ring networks are an interesting alternative since PMEs can be placed in independent smaller rings connected to external elements through Inter-Ring Routers (*IRR*) (see Figure 2(b)). This way local intra-ring communications do not suffer interferences from other rings. This section describes how single ring networks are modified so that they can be used in the context of multi-ring networks.

4.1 Controlled Injection-Rate Ring

One IRR handles all communications across two rings, receiving the flits coming from a source ring (*sR*) and sending them to the destination ring (*dR*). In the worst case, all nodes in the *sR* may want to send a flit to the *dR* through the same IRR, which means that the IRR can receive up to one flit per cycle from the *sR*. However, the IRR in the *dR* behaves as any other node and can inject flits at most once every N_{dR} cycles, where N_{dR} is the total number of nodes in the *dR*. Hence, the IRR may receive flits at a frequency higher than that at which it can send them to their destination in the *dR*. Thus, further injection rate control is needed to keep buffering requirements in the IRR boundable and low for the sake of hardware efficiency.

IRRs are equipped with two buffers to keep the flits from each of the two rings they connect until they are sent to the corresponding *dR*. Each of these buffers is, however, limited in size. In particular, each buffer has $N_{sR} - 1$ entries so that it can hold up to one flit from each node in the *sR* (except the IRR itself that cannot send flits to itself). By properly controlling the injection of each node, we guarantee that a flit from a node i in a ring *sR* sent to a node in the *dR* will not reach the IRR until the previous flit from such node to the *dR* has not been sent. This is done while still preserving the properties of the single rings: no stalls occur during traversal and buffering capabilities in nodes other than the IRRs are limited to store the incoming flit and forward it right away.

MFII. Unlike single rings, multi-rings have two types of MFII: (i) the $MFII_{sR}$ for flits whose final destination node belongs to the same ring as the sender node, and (ii) the $MFII_{sRdR}$ for flits whose final destination node belongs to another ring. The $MFII_{sR}$ is analogous to the single-ring case so that a node can send a flit only if N_{sR} cycles have elapsed since the previous flit was sent. Hence, $MFII_{sR} = N_{sR}$. Note that the previous flit may have been sent to a node in the *sR* or in the *dR*. Also, note that N_{sR} includes IRR nodes.

Communications to remote rings can occur less frequently than those to the local ring because they have to compete with local and remote network traffic. In particular, the IRR may need to wait N_{dR} cycles before injecting a flit into the *dR*. Thus, a flit sent by a particular node i in *sR* may remain up to $(N_{sR} - 1) \times N_{dR} - (N_{sR} - 2)$ cycles in the IRR if it reaches the IRR right after a flit from each other PME node in the *sR* has reached the IRR. If the flit from node i reaches the IRR when all other PME nodes' flits (up to $(N_{sR} - 2)$

flits) have just reached the IRR, it may need to wait for all those flits to be injected in the destination ring (up to N_{dR} cycles per flit and $(N_{sR}-2)$ flits), plus its own injection (N_{dR} cycles), so $(N_{sR}-1)\times N_{dR}$ cycles in total. However, those flits arrived at a rate of one flit per cycle at most, during the previous $N_{sR}-2$ cycles to the time when the flit from node i reached the IRR. Thus, if those flits are still in the IRR, the oldest one has been waiting at least for $N_{sR}-2$ cycles, so we must subtract this value from the total $MFII_{sRdR}$. Hence, $MFII_{sRdR} = (N_{sR}-1)\times N_{dR} - (N_{sR}-2)$.

Therefore, each node tracks (i) whether $MFII_{sR}$ cycles have elapsed since the last flit injection to allow a new flit to be sent to any local node and (ii) whether $MFII_{sRdR}$ cycles have elapsed since the last flit injection to a remote node to allow a new flit to be sent to any remote node.

WCTT. The WCTT for flits sent to the *sR* ($WCTT_{CIR}^{sR}$) is obtained as for the single ring. The WCTT for flits sent to the *dR* ($WCTT_{CIR}^{sRdR}$) is as follows:

$$\begin{aligned} WD_{CIR}^{sRdR} &= N_{flts} \times WD_{injflt}^{sRdR} \\ &+ WD_{injflt}^{dR} + WD_{trav}^{sR} + WD_{trav}^{dR} \end{aligned} \quad (6)$$

where WD_{injflt}^{sRdR} stands for the worst delay between flit injections whose destination is the remote ring, WD_{injflt}^{dR} is the worst delay that a flit in the IRR may take to be injected in the remote ring, and WD_{tran}^{sR} and WD_{tran}^{dR} stand for the worst traversal time in the local and remote rings respectively.

- WD_{trav}^{sR} and WD_{trav}^{dR} are obtained as for the single ring: $\left(D_{trav}^{router} + D_{trav}^{link}\right) \times H_d^s$. H_d^s is the number of hops between the source node and the IRR for WD_{trav}^{sR} and the number of nodes between the IRR and the destination node for WD_{trav}^{dR}.
- WD_{injflt}^{sRdR}: A flit sent to a remote node can be sent after $MFII_{sRdR}$ cycles have elapsed since the last flit was sent. Then, it may be the case that such flit has to wait for $N_{sR}-1$ cycles before being injected in the sR. Thus, WD_{injflt}^{sRdR} is obtained as follows: $WD_{injflt}^{sRdR} = MFII_{sRdR} + N_{sR} - 1$.
- WD_{injflt}^{dR}: Once a flit reaches the IRR it may have to wait for some time before it is injected in the *dR*. In particular, up to $MFII_{dR}$ cycles, so $WD_{injflt}^{dR} = MFII_{dR}$.
- N_{flts}: the number of flits per transaction is obtained as for the single ring. However, (i) $Link_{width}$ is either the same for both rings or is the lowest link width across the two networks; and (ii) $Header_{bits}$ may differ for local and remote communications.

MWC and MGC. MWC and MGC are achieved when each PME node sends continuously $N_{dR}-1$ flits to local nodes followed by one flit to a remote node. This allows all nodes in all rings (including the IRR) to be ready to send a new flit at any time. Thus, the behavior of all rings is analogous to that of single rings, so MWC and MGC are obtained as for single rings.

4.2 Rotating TDMA (*rtdma*) Ring

In this case, unlike to the case of single-ring networks, some slots must be reserved for local communications and other slots for remote communications. In particular, since a node can send a flit every N_{sR} cycles and a flit to a remote node every $(N_{sR}-1)\times N_{dR} - (N_{sR}-2)$ cycles, one every N_{dR} slots is reserved for remote communications ($N_{dR} \geq \frac{(N_{sR}-1)\times N_{dR}-(N_{sR}-2)}{N_{sR}-1}$).

The slot devoted to remote communications can be also used for local ones since local communications use a subset of the resources of remote ones. Conversely, slots reserved for local communications cannot be used for global communications unless enough time has elapsed since the last remote communication (at least $(N_{sR}-1)\times N_{dR}-(N_{sR}-2)$ cycles).

MFII. Under this setup $MFII_{sR} = N_{sR}$ for local communications since they can always use the next slot regardless of whether remote communications are allowed or not in that slot. For remote communications $MFII_{sRdR} = N_{sR} \times N_{dR}$ since only up to one every N_{dR} slots is reserved for those communications.

WCTT: The WCTT is obtained as for the *cir* multi-ring, but using those $MFII_{sR}$ and $MFII_{sRdR}$ for *rtdma* multi-ring networks to compute WD_{injflt}^{dR} and WD_{injflt}^{sRdR}. This is analogous to the case of single rings.

MWC and MGC. If all PME nodes are ready to inject a continuous sequence of $N_{dR}-1$ consecutive flits for local communications followed by one flit for remote communications, then MGC and MWC are 1 analogously to single-ring networks.

5. WCET AND HW/SW SUPPORT

Our timing-analyzable ring-based network can be easily used in the context of Static Timing Analysis (STA). STA tools need to know the latencies of the different hardware resources at analysis time to derive WCET estimates statically. Our approach provides the WCTT for each ring traversal. This value can be used trivially to increase the latency of those resources that must be accessed through the ring. For instance, if a core issues a load operation to memory, memory latency (WD_{load}) must be increased by the WCTT to reach memory ($WCTT_{to-mem}$) and by the WCTT to get the data back ($WCTT_{to-core}$), so that the new load latency would be as follows:

$$WD_{load}\prime = WCTT_{to-mem} + WD_{load} + WCTT_{to-core} \quad (7)$$

where $WCTT_{to-mem}$ and $WCTT_{to-core}$ are obtained analogously to $WCTT_{CIR}$ with the appropriate hardware parameters. Thus, STA tools do not need to be changed in any way. Instead, the proper latencies for the different components must be provided considering the WCTT of the ring-based network.

For measurement-based timing analysis (MBTA) tools, we use an existing hardware feature, the WCET Computation Mode [11], which is enabled at analysis time and enforces each request to a given resource to experience its worst-case delay. In the particular case of our ring, the program under study is run in isolation, and its ring accesses are enforced to wait for $WCTT_{CIR}$ cycles before being effectively injected. Therefore, the worst potential inter-task interferences are experienced at analysis time, so WCET estimates are trustworthy and no change is needed in the MBTA tools.

5.1 Controlled injection Rate Ring

The controlled injection rate ring network only needs PME to be extended with hardware avoiding the injection of requests in less than N_{sR} cycles since the former injection. Note that such control can be trivially implemented and does not require any coordination with other PME, since each PME decides when it can allow the next flit to be injected based only on its own past history.

Multi-rings require the same support as single rings for timing analysis. The only difference is that multi-rings require all nodes to be extended with hardware avoiding the injection of requests in less than $MFII_{sR}$ cycles since the last injection for local communications, and less than $MFII_{sRdR}$ cycles for remote communications. The IRR is required to have two buffers (one per ring) with as many entries as nodes in the source ring for the buffer. Each entry must have the same number of bits as a flit.

5.2 rTDMA

The same methods used for STA and MBTA in the presence of the controlled injection rate ring can be used in the presence of the *rtdma* ring. The only difference is the fact

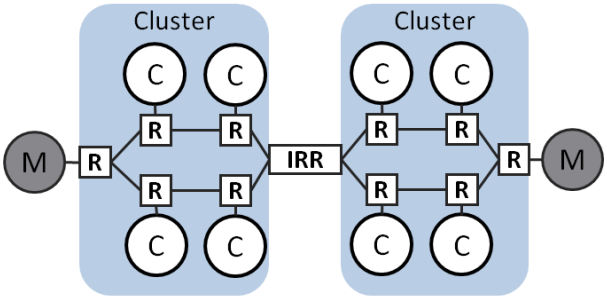


Figure 5: Multi-ring setup evaluated in this paper. The single-ring setup comprises only one of the clusters. 'C' stands for core, 'R' for router and 'M' for memory.

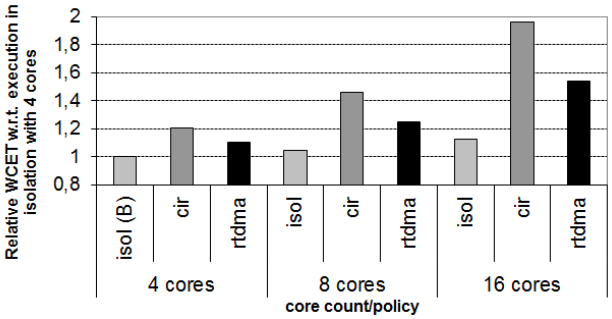


Figure 6: WCET single-ring

that WCTT values will differ. In summary, the *rtdma* ring requires global synchronization to guarantee that all nodes inject their flits simultaneously. If such a hardware support is in place, this design is the best choice since its WCTT is lower. However, achieving global synchronization may be non-trivial to implement.

Note that the same synchronization support as for *rtdma* single rings is required for *rtdma* multi-rings.

5.3 Prioritization

In this paper, we assume that all task contending for access to the ring have the same priority. However, there may be scenarios in which prioritizing one task over the other is desirable such as in *mixed critical workloads*, in which, although a set of applications has less priority than others, providing a WCET estimation is still mandatory.

Prioritization introduces two key issues. On one hand, it has to be provided in such an a way that WCTT can be still computed since as mentioned in Section 2, a simple prioritization of either injecting or transient traffic makes WCTT potentially impossible to bind. On the other hand, *time composability* is affected by prioritization because tasks must know which tasks have higher priority to determine its own WCTT for each request as lower priority requests are stalled by higher priority requests. This last issue is left to the system designer who should balance the benefits offered by Time Composability as shown in Section 2, and the benefits of prioritization.

Both ring approaches presented in this paper allow priority arbitration policies. In the case of the controlled-injection rate approach, we can simply vary the MFII for each task, increasing the MFII for low priority tasks and decreasing the MFII of high-priority tasks. Note that to do so, the number of high priority requests that can affect a low priority request must be known. Moreover, this has to be done keeping the no-preemption principle, i.e. preventing that when requests start traversing the ring it suffers no inter-task delays. In case of rTDMA, priority arbitration policy can be implemented by increasing the window size given to high-priority tasks, and so reducing the window size of low-priority tasks, so high-priority tasks have more chances to get access to the bus.

6. RESULTS

Setup. All results presented in this paper has been obtained with a power-pc binary-compatible execution-driven simulator based on SoCLib simulation framework [22]. The simulator models a 2-stage processor with 4KB 8-way 32B/line first level separated instruction (IL1) and data (DL1) caches with write-back policy to main memory. IL1 and DL1 hit latency is 1 cycle. In case of a miss, a memory request is sent through the ring network. The simulator also models two ring setups: A single-ring setup with one memory device and core counts varying from 4 to 16; and a clusterized architecture a-la ARM Cortex A15 MPCore [23], deploying a multi-ring setup with 2 rings (see Figure 5), each with one memory device and a core count varying from 4 to 8. We consider that link and router traversal times are 1 cycle. The link bandwidth matches header width plus 64 bits for data/address. At every load operation missing in the IL1 or DL1, a packet of 1 flit with the address is sent to memory; the memory response contains 4 flits of data. Every write operation missing in DL1 sends a 4-flit packet to memory, which answers with a 1-flit packet on completion. Finally, and without loss of generality, the simulator models on-chip memory(ies), similar to the GR712RC [24]. In case their memory is off-chip this will simply introduce an extra-delay in each request [12] but without any change on the principles of the CIR and rTDMA ring designs.

The off-chip communication, i.e. the communicating among different multicore chips, is out of the scope of this paper. In this case, other network technologies are used, such as FlexRay or CAN (controller area network) buses are used in the automotive domain and AFDX (Avionics Full-Duplex Switched Ethernet) in the avionics domain.

Benchmarks. The most common way of exploiting multicores in current CRTES is by aggregating (consolidating) single-threaded tasks onto the same chip. Communication happens mainly at task boundary, and its duration is relatively small in comparison with the task duration. There are several proposals trying to derive WCET bonds for truly parallel tasks [25], but this work is still in progress and will take years to be adopted by industry. In this paper, we use a benchmark suite composed of single-threaded tasks. In particular EEMBC Autobench benchmark suite [26], which is widely used as a representative of some automotive applications.

WCET estimates. We obtain WCET estimates using RapiTime [10], a commercial tool developed by Rapita Systems Ltd. widely used in the avionics, telecommunications, space and automotive industries. RapiTime estimates the WCET using a measurement-based technique and uses on-line testing to measure the execution time of sub-paths between instrumentation points in the code. RapiTime, therefore, uses path analysis techniques to build up a precise model of the overall code structure and determine which combinations of subpaths form complete and feasible paths through the code. Finally, RapiTime combines the measurement and control how analysis information to compute measurement based worst-case execution time estimations in a way that captures accurately the execution time variation on individual paths due to hardware effects.

6.1 WCET Estimates

6.1.1 Single-ring Setup

Figure 6 shows the normalized WCET for all single-ring setups under control-injection rate (*cir*) and rotating TDMA (*rtdma*) for 3 core counts 4, 8 and 16. For comparison purposes, we also show the WCET when applications run in isolation (*isol*), i.e. with no injection control $MFII = 0$,

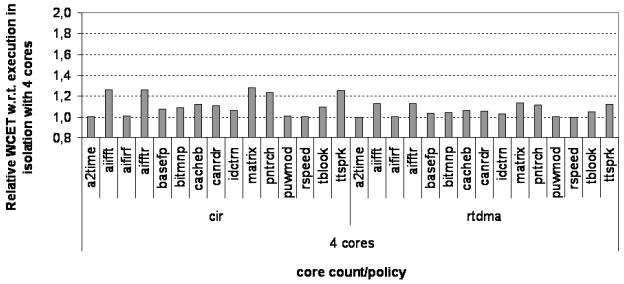


Figure 7: Per-benchmark WCET results under the 4-core single-ring setup

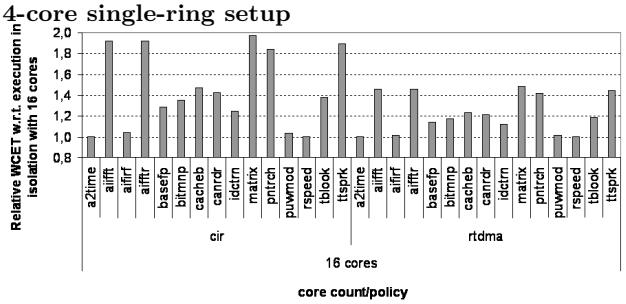


Figure 8: Per-benchmark WCET results under the 16-core single-ring setup

under each core count. All WCET estimates are normalized to WCET under the 4-core*isol* setup.

In *isol* the effect of increasing the number of nodes in the ring is low. The increased main memory latency due to the extra nodes in the ring produces a relatively low WCET increase. This is so because the ring delay is at least one order of magnitude lower than the memory latency. In particular, WCET degrades by 4% when moving from 4 to 8 cores and by 14% when moving from 4 to 16.

As expected, *rtdma* provides tighter WCET estimates than *cir*. This is so because the WCTT for *rtdma* is smaller than for *cir* in all the evaluated setups. WCET is 9%, 14% and 21% smaller for *rtdma* than for *cir* for the 4-, 8- and 16-core setups. However, recall that *rtdma* requires global synchronization, which has been shown to introduce complex implementation issues due to the complexity and power dissipation of strategies to mitigate clock skew [21]. It is up to the hardware designer to choose the best balance between WCET tightness and hardware cost.

Figure 6 also allows us assessing the effect of using one of the conflict-aware techniques presented in this paper, *cir* and *rtdma*. In particular, for a given core-count the effect of using *rtdma* or *cir* with respect to the WCET estimate in isolation is small. WCET estimates degrade by 10%, 20% and 37% for *rtdma* for 4-, 8- and 16-core setups. WCET degradation is 19%, 41% and 74% for *cir*.

Per-benchmark results for 4- and 16-core count are depicted in figures 7 and 8 respectively. Results for 8-core follow the same trend and are omitted for space constraints. In those figures, we show the normalized WCET for all single-ring setups under control-injection rate (*cir*) and rotating TDMA (*rtdma*). We normalize WCET estimates to the case when each application runs in isolation in a setup with the same core count.

The effect varies significantly across benchmarks. The effect of the ring setup on a benchmark depends on how often it accesses the ring. This is, in turn, determined by the density of cache misses of the benchmark. The benchmarks with higher sensitivity across all setups are those with higher miss rates: *aiifft*, *aifftr*, *matrix*, *pntrch* and *ttsprk*. The same conclusions we obtain in Figure 6 hold for individual benchmarks: 1) When benchmarks run in *isol*, the effect of increasing the number of nodes in the ring is limited. 2) *rtdma* provides tighter WCET estimates than *cir*

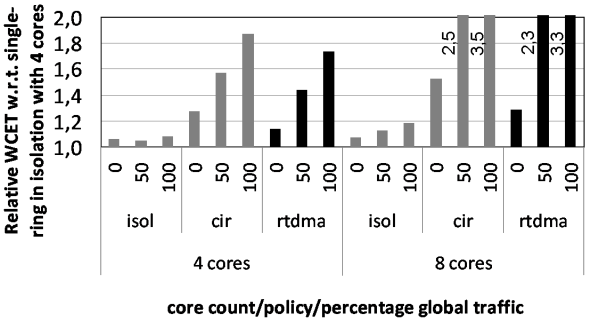


Figure 9: WCET multi-ring

because *rtdma* leads to smaller WCTT values. 3) The effect of using *cir* and *rtdma* with respect to the WCET estimate in *isol* is relatively small.

6.1.2 Multi-ring Setup

Figure 9 shows the normalized WCET for each multi-ring setup (4 and 8 cores per ring). WCET estimates are normalized to *isol* when applications run in a 4-core single-ring setup. We show WCET estimates under 3 scenarios in which we vary the fraction of data/instructions that applications have in the memory device at the remote ring. *0* means that 0% of the data is in the remote memory element (so 100% of the data resides in the local memory).

The fraction of global traffic plays the most important role in the WCET estimation. In the 4-core per-ring setup, WCET degrades 26% and 52% when the percentage of global traffic increases from 0% to 50% and 100% respectively. In the 8-core per-ring setup WCET increases by 78% and 156% respectively. Mapping applications close to their data (or the mapping of data into memory modules) is critical for the tightness of WCET estimates and is part of our future work.

Again WCET estimates obtained for *rtdma* are better than for *cir* since WCTT values are lower for *rtdma* though the difference is smaller than for single rings. In the case of the setup with 4 cores per ring *rtdma* improves over *cir* by 10%, 8% and 7% for global traffic rates of 0%, 50% and 100% respectively. In the case of 8 cores per rings differences are similar: 15%, 9% and 7%.

Per-benchmark results for the multi-ring setup show exactly the same trend as in Figure 9, so they have been omitted.

We can also compare the single ring (Figure 6) versus the multi-ring (Figure 9) setup. For the 8-core count, if all traffic is local (0% global traffic) the multi-ring setup leads to tighter WCET estimates than the single ring, 13% and 8% lower under *cir* and *rtdma* respectively. However, if the percentage of global traffic is 50% then the multi-ring is worse in terms of WCET than the single-ring (8% and 16% respectively for *cir* and *rtdma*). This result reinforces our point on the importance of a smart mapping of applications close to their data, hence reducing global traffic as much as possible.

6.2 Average performance

Figure 10 shows the average performance obtained for *cir* and *rtdma* under the 4-core single ring case. We also show the case in which no contention-aware policy is used. That is, a setup in which no WCTT can be provided but is aimed at improving performance. For this experiment, we run 16 4-EEMBC randomly generated workloads. The particular workloads are shown in Table 1. We measure performance as the IPC throughput of all benchmarks in a workload, i.e. we sum the instructions per cycle (IPC) of every benchmark in a workload.

Both *cir* and *rtdma* obtain average performance *very close*

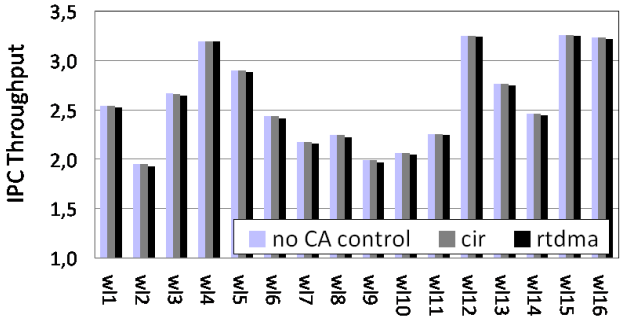


Figure 10: Average performance for the 4-core single-ring case

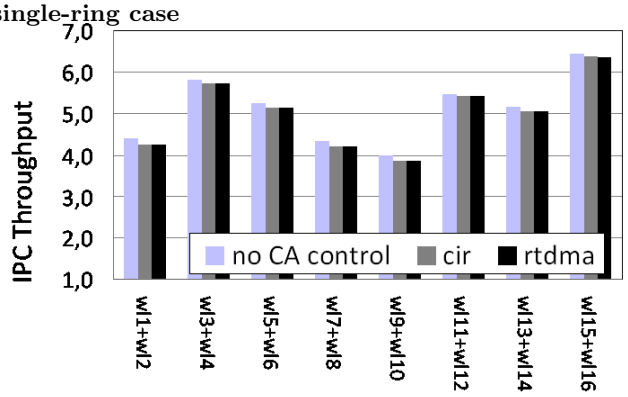


Figure 11: Average performance for the 8-core single-ring case

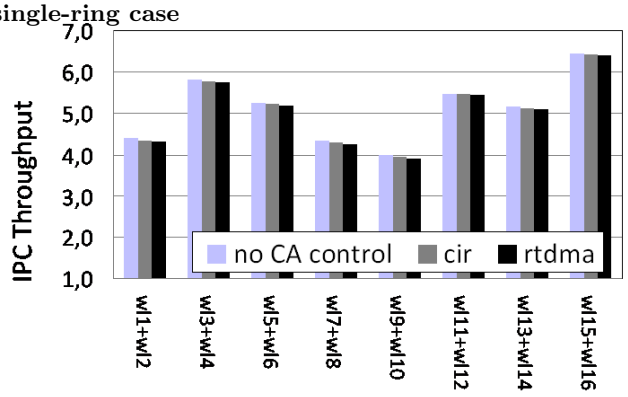


Figure 12: Average performance for the 8-core multi-ring case (2 rings with 4 cores each)

to the performance when no contention-aware policy is used. This is so because the time between two accesses to the ring, i.e. the time between any read miss in the data or instruction caches and write operations, is usually longer than the MFII, so there is almost no slowdown when using *cir*. There is a slight slowdown in the case of *rtdma* because once an access is ready, it has to wait to its task slot in the TDMA to be sent. Overall, we observe that it is possible to obtain both high average performance and bounded WCTT.

Setups with 8 cores have also been analyzed. In particular, we analyze the case where the 8 cores are arranged in a single ring (see Figure 11) and where they are arranged in 2 rings with 4 cores each (see Figure 12). Workloads for the 8-core count have been generated by composing 4-core workloads arbitrarily: workload 1 for 8 core setup consists of workloads 1 and 2 for 4 core setup, workload 2 of workloads 3 and 4 and so on. Since 4-core workloads have been randomly generated, 8-core workloads are also random.

Slowdowns of *cir* and *rtdma* with respect to the no contention aware policy grow a bit in the single ring case due to the increased core count that increases *MFII*. Thus, some more accesses get delayed few cycles due to a previous access recently sent. For the multi-ring, we observe that the difference between *cir* and the case in which there is no contention

Table 1: 4-task workloads

workload id	task 1	task 2	task 3	task 4
wld-4c-01	tblook	tblook	idctrn	pntrch
wld-4c-02	ttsprk	idctrn	cacheb	aifftr
wld-4c-03	pntrch	canrdr	rspeed	canrdr
wld-4c-04	rspeed	rspeed	aifirf	aiifft
wld-4c-05	cacheb	a2time	aifirf	pntrch
wld-4c-06	pntrch	basefp	canrdr	idctrn
wld-4c-07	bitmnp	matrix	puwmod	pntrch
wld-4c-08	basefp	aifftr	tblook	canrdr
wld-4c-09	aifftr	pntrch	basefp	idctrn
wld-4c-10	tblook	puwmod	matrix	aifftr
wld-4c-11	matrix	a2time	puwmod	matrix
wld-4c-12	rspeed	rspeed	ttsprk	rspeed
wld-4c-13	aifftr	bitmnp	idctrn	a2time
wld-4c-14	basefp	aifirf	basefp	aiifft
wld-4c-15	rspeed	a2time	pntrch	puwmod
wld-4c-16	cacheb	puwmod	puwmod	cacheb

aware policy in place is almost null. This is so because the time between two consecutive accesses to the ring, i.e. the time between any read miss in the data or instruction caches and write operations, is usually longer than the *MFII* (50% of the misses are randomly sent to the remote ring while the rest are kept local). Hence, a given request is not delayed due to the fact that the last request was sent more than *MFII* cycles before.

The slowdown for *rtdma* is a bit higher, 3% in the worst case. This slowdown in is due to the fact that once a request to the ring is ready, it has to wait to its task slot in the TDMA to be sent.

Overall, these results confirm that it is possible to obtain both high average performance and bounded WCTT also for 8-core counts, which makes these two policies attractive for CRTES.

7. RELATED WORK

Communication across chips in CRTES has been typically implemented with technologies such as FlexRay [27] and CAN [28] in the automotive domain, or AFDX [29] in the avionics domain. However, the need for higher performance and integration makes multicores a must in CRTES. Several network designs exist to connect nodes in a multicore processor, including meshes and buses, rings have been shown to provide good implementation cost and performance ratio which has made several chip designers implement them recently [2][3][4][5].

Note that the requirement for Rings in CRTES is not providing an average guaranteed bandwidth allocation for the program, but to provide an upper bound to the latency of every single request to the ring. Rings and particularly bidirectional rings have been thoroughly analyzed in terms of throughput and fairness [14, 15]. QoS techniques, even if very varied and suited to different traffic scenarios [16, 17], do not attempt to provide an upper bound for the delays, but best effort tradeoff, to reconcile (i) obtaining the maximum throughput possible while (ii) still achieving fairness and providing the capability of prioritized traffic. In the CRTE domain, simplicity, cost and analyzability are not only desirable features, but requirements that differentiate our work form previous ones.

The literature on interconnection networks providing hard real-time guarantees such as the WCTT covers buses [11] [30], switched on-chip networks such as meshes [31, 32, 33], rings [31] and even application specific network on-chip designs [34] in which associated CAD tools are used to create platforms based on use-cases and task-graphs.

It is of special interest [31], in which authors proposed a time-division-multiplexed (TDM) arbitration policies for on-chip rings in which access requests are *statically scheduled* to avoid collisions. Statically scheduling requests through software when the communication happens at a time scale of microseconds or milliseconds and when connecting components that are visible to the software such as an I/O device or a DMA. However, the communication that happens on a chip level (such as access to memory/cache) is much more frequent (in the order of nanoseconds) and hence are much harder if at all possible, to be statically scheduled at the software level to prevent collisions among tasks. Our ring designs builds on the fact that the load that running tasks

put on the ring is hard to be taken into account when computing the WCET for a task. Instead, our design provides time-composable WCTT making the WCET of a task independent of the load put on the ring. This, in addition to providing the benefits brought by time composable designs on reducing timing validation and verification costs, does not introduce any change on the WCET analysis tool to work with ring-based architecture, since the WCTT can be easily taken into account by the tool.

8. CONCLUSIONS AND FUTURE WORK

Ring networks, both single and multi-ring, have been extensively analyzed in the context of parallel computers, leading to a rich area of research with numerous proposals to improve performance, scalability and utilization, and to reduce cost and energy. However, in CRTES, even these simple networks have not been thoroughly designed and analyzed to provide WCTT bounds. First, to be used in CRTES, the design must be extremely simple, for cost and energy (as they will be part of embedded systems), but also to allow an exact calculation of the WCTT.

We propose several time-predictable single and multi-ring designs in shared-memory multicores providing (i) composable WCTT as required by WCET analysis in CRTES and (ii) low hardware complexity. We show with a commercial WCET tool on EEMBC benchmarks that the presented ring designs successfully meet both objectives. In both cases, the WCET estimates obtained are not far from the observed average execution times of applications in the ring.

Our results, though constrained to CRTES, might also be applicable to large-scale networks, where unbounded end-to-end delays in addition to jitter are starting to lead to huge performance penalties in collective communication patterns [35].

9. ACKNOWLEDGMENTS

The research leading to these results has been funded by the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA). This work has also been funded by the Ministry of Science and Technology of Spain under contract TIN2012-34557. Eduardo Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the grant Juan de la Cierva JCI2009-05455. Miloš Panić is funded by the Spanish Ministry of Education under the FPU grant FPU12/05966.

10. REFERENCES

[1] T. Lundqvist and P. Stenstrom, "Timing anomalies in dynamically scheduled microprocessors," in *RTSS*, 1999.

[2] R. Riedlinger et al., "A 32nm 3.1 billion transistor 12-wide-issue Itanium processor for mission critical servers," in *ISSCC*, 2011.

[3] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM J. Res. Dev.*, vol. 46, no. 1, 2002.

[4] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, "POWER5 system microarchitecture," *IBM J. Res. Dev.*, vol. 49, no. 4/5, 2005.

[5] "Unleashing the cell broadband engine processor.http://www-128.ibm.com/developerworks/power/library/pa-fpfeib/," 2005.

[6] "Platform 2015: Intel processor and platform evolution for the next decade."

[7] G. Ravindran and M. Stumm, "A performance comparison of hierarchical ring- and mesh- connected multiprocessor networks," *HPCA*, 1997.

[8] C. Fallin, X. Yu, G. Nazario, and O. Mutlu, "A high-performance hierarchical ring on-chip interconnect with low-cost routers," Computer Architecture Lab, Carnegie Mellon Univ, Tech.Rep. 2011-007, 2011.

[9] V. C. Hamacher, H. Jiang, V. C. Hamacher, S. Member, and H. Jiang, "Hierarchical ring network configuration and performance modeling," *IEEE Transactions on Computers*, vol. 50, 2001.

[10] RapiTime, *www.rapitasystems.com*, 2008.

[11] M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for WCET analysis of hard real-time multicore systems," in *ISCA*, 2009.

[12] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, *An Analyzable Memory Controller for Hard Real-Time CMPs*, IEEE Embedded Systems Letters, 2009.

[13] M.-K. Yoon, J.-E. Kim, and L. Sha, "Optimizing tunable WCET with shared resource allocation and arbitration in hard real-time multicore systems," in *RTSS*, 2011.

[14] I. Cidon and Y. Ofek, "Metaring-a full-duplex ring with fairness and spatial reuse," in *INFOCOM*, 1990.

[15] S. Gjessing and A. Maus, "A fairness algorithm for high-speed networks based on a resilient packet ring architecture," in *International Conference on Systems, Man and Cybernetics*, vol. 2, oct. 2002.

[16] J. Pelissier, "Providing quality of service over infiniband architecture fabrics," in *8th Symposium on Hot Interconnects*, 2000.

[17] F. J. Alfaro, J. L. Sanchez, and J. Duato, "Studying the influence of the infiniband packet size to guarantee qos." in *ISCC*, 2005.

[18] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: M.K. Publishers Inc., 2003.

[19] C. Carrión, R. Beivide, J. A. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," ser. HiPC, 1997.

[20] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. M. Prellezo, "The adaptive bubble router," *JPDC*, vol. 61, 2001.

[21] D. Harris and S. Naffziger, "Statistical clock skew modeling with data delay variations," *IEEE Transactions on VLSI Systems*, vol. 9, 2001.

[22] SoCLib, http://www.soclib.fr/trac/dev.

[23] ARM Ltd., "Cortex-A15 Processor," in *http://www.arm.com/products/processors/cortex-a/cortex-a15.php*.

[24] Aeroflex Gaisler, "GR712RC dual-core LEON3FT SPARC V8 Processor," in *http://www.gaisler.com/index.php/products/components/gr712rc*.

[25] parMERASA, *EU-FP7 Project: http://www.parmerasa.eu/*.

[26] J. Poovey, *Characterization of the EEMBC Benchmark Suite*, North Carolina State University, 2007.

[27] FlexRay Consortium, "FlexRay communications system. protocol specification V2.1 Rev. A," 2005.

[28] International Organization for Standardization, *ISO 11898. Road vehicles – Controller area network (CAN)*, 2003.

[29] ARINC, *Specification 664 Part 7 – Avionics Full DupleX Switched Ethernet (AFDX)*, Aeronautical Radio, Inc, 2005.

[30] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Bus-aware multicore wcet analysis through tdma offset bounds," *ECRTS*, 2011.

[31] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *NoCs*, 2012.

[32] S. Ward et al., "The numesh: a modular, scalable communications substrate," in *ICS*, 1993.

[33] M. Schoeberl, "A time-triggered network-on-chip," in *FPL*, 2007.

[34] A. Hansson, M. Subburaman, and K. Goossens, "aelite: a flit-synchronous network on chip with composable and predictable services," in *DATE 2009*, pp. 250–255.

[35] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation," in *SC*, 2010.