CORE
Metadata, citation and similar papers at core.ac.uk
Provided by UPCommons. Portal del coneixement obert de la UPC

EOmesh: Combined Flow Balancing and Deterministic Routing for Reduced WCET Estimates in Embedded Real-Time Systems

Jordi Cardona, *Universitat Politècnica de Catalunya (UPC) and BSC*
Jaume Abella and Carles Hernandez, *Barcelona Supercomputing Center (BSC)*
Francisco J. Cazorla, *BSC and IIIA-CSIC*

***Abstract*—The increasing performance needs in critical real-time embedded systems (CRTES) can only be satisfied with the use of high-performance manycore processors. While NoC-based manycore systems are popular in the high-performance domain due to their high average performance, they challenge deriving tight Worst-Case Execution Time (WCET) estimates, as needed in CRTES. Weighted meshes have been proposed to alleviate NoCs pathological behavior – caused by large bandwidth imbalance – by making locally unbalanced arbitration decisions to reach globally balanced bandwidth. In this paper we show that existing weighted mesh solutions do not completely remove unwanted imbalance, in particular for nodes subject to high congestion. We propose *EOmesh*, an approach that combines heterogeneous predictable routing and weight allocations that delivers near-optimal bandwidth allocation across cores without increasing NoC complexity. EOmesh, which can be implemented either by hardware means or by software means on top of regular weighted meshes, improves the average performance and WCET results of the reference weighted mesh design.**

***Index Terms*—NoC, Weighted Mesh, Bandwidth, Latency.**

I. Introduction

The relentless need for increasing levels of computing performance in critical real-time embedded systems (CRTES), such as automotive [2], calls for the adoption of high-performance hardware as the only effective way to deliver such performance. However, complex hardware designs confronts with CRTES need to undergo strict validation and verification processes in accordance with functional safety regulations such as ISO26262 in the automotive domain [6] and DO178B/C in the avionics domain [24]. In particular, complex hardware designs hampers deriving evidence on the timely execution of critical real-time tasks, typically in the form of Worst-Case Execution Time (WCET) estimates.

Deriving WCET estimates (bounds) requires the use of hardware/software platforms on which execution time can be upper-bounded. Further, those bounds must be low enough so that they are useful (e.g. guaranteeing that the braking system of a car would take no more than 5 seconds to react is useless in practice). In general, this translates into using only a subset of the existing performance-improving hardware designs and configurations, despite the negative impact on average performance of this constraint.

NoC-connected manycores have already been deployed in high-performance (mainstream) processors [22], [30], and have been shown to allow – under some restrictions, e.g. deterministic routing – the derivation of tasks' WCET estimates as needed for CRTES [16], [20]. However, WCET estimates vary drastically across cores in mesh NoCs due to the different bandwidth effectively allocated to each core and, to a lower extent, the diverse latencies caused by non-homogeneous distances from cores to their target node (e.g. the one where main memory is attached) [15]. Local homogeneous bandwidth allocation (e.g. as provided by fair policies like round-robin) might cause unwanted heterogeneous global bandwidth distribution across cores (i.e. cores further away from memory have lower bandwidth than those closer to it). As a result, tasks running in cores with lower allocated bandwidth can be severely penalized. Parallel applications, despite less widespread than in the mainstream market, are now considered in CRTES for computing intensive functions related to autonomous driving and unmanned navigation. For those applications, unwanted bandwidth imbalance may result in poor WCET estimates since all threads may have to synchronize with the slowest one (i.e. normally the thread that runs in the farthest core from memory).

Weighted meshes, widely used in high-performance routers for off-chip wormhole networks [3], have been proposed as a solution in CRTES to allow heterogeneous bandwidth allocation across ports to homogenize overall bandwidth across cores [15]. Unfortunately, weighted meshes suffer from a key limitation: *efficiently-sized resources in NoCs may create bubbles in some links, so that nodes are unable to send packets sustainedly. For scenarios with highly unbalanced bandwidth distribution in some routers, bubbles may prevent from reaching the desired bandwidth allocation in those routers, leading to unwanted globally unbalanced bandwidth allocations.*

In this paper we tackle this challenge by proposing EOmesh, a new weighted mesh design that effectively achieves near-optimal (homogeneous) bandwidth allocation across cores. EOmesh builds on the observation that globally-fair bandwidth allocation can only be practically achieved by combining the use of weights in the arbiters and balancing the amount of flows served at each output port. In particular, our contribu-

This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis 2018 and appears as part of the ESWEEK-TRAD special issue. Permission to make digital or hard copies of part of all of this work for personal or classroom use in granted without fee provided that copies are not made or distributed on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

tions are as follows:

1) We analyze the behavior of weighted meshes, showing that, by construction, they cannot achieve homogeneous bandwidth allocation across cores as long as one port needs to serve above $2/3$ of the incoming requests, which necessarily occurs for square meshes with 4x4 nodes and beyond.
2) We provide a smart combination of time-predictable routing policies (XY and YX) that allows reducing the imbalance across ports so that bubbles do not prevent from reaching the desired homogeneous bandwidth allocation. The combination of multiple routing policies with statically allocated virtual channels makes the mesh NoC deadlock-free.
3) We use appropriate weight allocations that achieve homogeneous bandwidth allocation across cores, in combination with the new routing scheme. Hence, fairness across cores in terms of bandwidth is achieved despite bubbles in the NoC.

A key characteristic of EOmesh is that it departs from real-time specific NoCs that, for instance, add new signals among routers and nodes, different flow-control, and global clocks. Instead, EOmesh targets high-performance COTS wormhole NoCs (wNoCs). Hence, EOmesh achieves reduced WCETs by leveraging an optimal configuration of wNoCs parameters, e.g. arbitration and routing that are already configurable by software in existing real NoCs. Interestingly, while weight allocation is not in commercial NoCs, its low implementation costs, just requiring arbiter-local changes, makes it a candidate for future NoCs.

In terms of evaluation, we provide implementation details of our design, EOmesh, showing that it only requires changes in the routing choices and weight allocation, both of which can, in general, be programmed by software. We also assess the performance of EOmesh showing that it outperforms non-weighted and weighted mesh designs in terms of both, average performance and WCET. WCET gains come from the fact that EOmesh routing scheme prevents some flows from colliding in many routers by construction. Moreover, as part of our evaluation, we assess, for the first time, the impact of these NoC designs in the context of critical real-time parallel applications.

The rest of the paper is organized as follows. Section II provides some background on timing analysis and NoCs. Section III presents a detailed model of weighted meshes and analyzes their limitations. Section IV introduces our new design, the EOmesh. Section V provides implementation details of the weighted meshes in general and the EOmesh in particular. Evaluation results are provided in Section VI. Some related work and conclusions are given in Sections VII and VIII respectively.

II. Background

This section provides some background on timing analysis practice, and the properties required from the platform used. It also presents the NoC designs considered in this work and the weighted mesh upon which we build our contributions.

A. Timing Analysis

Timing analysis contributes to the timing validation and verification process, mandatory in CRTES. Timing analysis derives evidence on how timing budgets assigned to each task suffice to guarantee the execution of those tasks. To that end, a WCET estimate for each task is derived. Different methods exist for WCET estimation, spanning from static timing analysis (STA) to measurement-based timing analysis (MBTA) [33]. Each paradigm is used based on practical and economical viability for the target application and hardware/software platform considered.

Due to economical reasons, WCET estimates are required early in the design process so that potential budget violations are detected and fixed at a low cost. Leaving such verification for late design stages may cause expensive recalls and jeopardize time-to-market. In this line, it is desirable that WCET estimates are time-composable, so that they hold valid across incremental integration of the different software components. Finally, another desirable property of WCET estimates is tightness, so as to enable an efficient use of resources by allowing the consolidation of as much software as possible onto a single hardware platform.

The estimation of time-composable WCET bounds requires accounting for the worst potential congestion that can occur upon integration of other tasks in multicores. In the context of STA and NoCs, this requires computing the worst (theoretical) contention that requests can experience to traverse the NoC from their source to their destination [4], [11]. Then, contention bounds are added to the intrinsic latency of requests in a contention-free scenario.

Two main approaches exist to estimate such worst contention: Worst-Case Traversal Time (WCTT) [20] and Worst-case Contention Delay (WCD) [16]. The former accounts for the worst overall contention that a request can experience, which may be caused (at least partially) by previous requests of the same task. Thus, an event causing contention on multiple requests of the same task would be accounted multiple times. In particular, one request experiences such contention and the other ones get stalled due to backpressure of the other request. The latter, WCD, accounts for the contention caused by requests of other tasks only, thus not accounting for backpressure caused by requests of the same task. This avoids accounting multiple times for the same contention, thus delivering tighter (yet reliable) WCET estimates. In fact, measurement-based analysis of worst-case contention scenarios reveal that WCD tightly upper-bounds maximum contention [16]. Hence, we build on WCD for WCET estimation in this paper. In particular, we estimate the WCD, which can be added to requests latency in the context of STA, or can be factored in WCET estimates obtained with MBTA by adding WCD cycles per request to the WCET estimate obtained in a contention-free scenario.

B. NoCs

NoCs comprise point-to-point links and routers to send packets across nodes. NoC characteristics impact the latency of requests to traverse them from source to destination. In

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

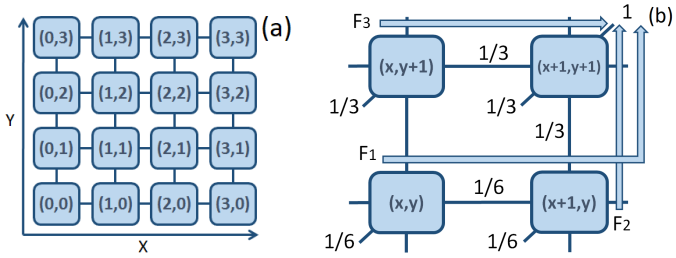


Fig. 1: Router coordinates (a); and unfair bandwidth allocation under wormhole (b).

this work we consider a mesh NoC, as it has been shown to deliver high performance and are implemented in several processors [30], [22]. Communication at the NoC level requires sending individual packets from source to destination. Those packets (payload and control information) may require higher bandwidth than that allowed by NoC links. E.g., 500-bit packets cannot be sent atomically through a 128-bit wide link. Hence, packets are split into flits (short of FLow control unITs), which can be sent atomically as they are not wider than links.

In the context of NoCs, wormhole switching is the most adopted approach due to its low buffering requirements. In a wormhole NoC every core request translates into a packet, which is the minimum arbitration unit. A packet can be split into one or several flits. The header flit of a packet contains the destination information required to forward the packet to the corresponding router output port. Once the header flit is granted access to a given output port, the remaining flits are forwarded to this port without any further arbitration.

WCET estimation is easier to carry out on deterministic routing policies. In particular, XY routing has been shown to be very suitable for meshes as WCD (and WCTT) bounds can be easily determined and its implementation complexity is very low [16], [20]. XY routing builds upon forwarding packets in the X direction first until reaching the column of the destination node, and then forwarding them in the Y direction, thus making routes to have minimal length (in terms of hops) and making routing decisions to be fully deterministic in any router (a single direction can be chosen for a given packet). In order to choose the packet to be granted access to an output port, we build upon round-robin arbitration, which has been shown to be a (locally) fair and easy to implement arbitration policy in NoCs [7]. However, for meshes, homogeneous round-robin at router level delivers highly unbalanced bandwidth across nodes since nodes closer to destination and those in routes with fewer flows receive higher overall bandwidth than the other cores [17].

C. Weighted Mesh

Weighted meshes [17] grant heterogeneous bandwidth in the routers to the different flows to achieve a globally-fair (homogeneous) bandwidth allocation across cores. This is achieved by using, for instance, a larger arbitration window in the case of round-robin, so that a larger number of slots is given to some ports so that the overall bandwidth allocated to each core to the destination is homogenized. Conceptually, given a NoC with *NxM* nodes, weighted meshes reduce the bandwidth for nodes whose bandwidth is above $\frac{1}{N \times M}$ and increases it for those whose bandwidth is below. Such an approach has been shown doable and suitable for WCD estimation in the context of critical real-time systems [15].

TABLE I: Definitions used in this paper.

Acronym	Description
$R(x,y)$	Node with coordinates (x,y) in the NoC
F_i	Flow i
P_i^j	Number of requests that might contend for the same R^j output port as F_i under the worst-case contention scenario
ER_i^j	Rate at which flits of flow F_i can be ejected from router R^j
D_i^j	Maximum time that a packet of F_i requires to go from the input port of R^j to its destination node
$fx\{i\}$	Index of the flow causing the worst possible blocking on F_i
PER^{wc}	Propagated worst-case ejection rate
$PER_i(R^j)$	Propagated worst-case ejection rate for flow F_i at router R_j
L	Maximum packet size
I_{dir}	Number of flows traversing the dir input port
O_{dir}	Number of flows traversing the dir output port
N_{ports}	Number of ports per router

III. Modeling a Weighted Mesh

WCET estimation in manycores needs bounding access times to shared hardware resources [14]. In the case of NoCs, this translates into having a bounded WCD such that every request sent to the NoC has a bounded service (traversal) time at analysis time.

A. Baseline NoC

To better understand the performance guarantees provided by weighted meshes, we model a canonical 2D wormhole mesh router comprising five input ports that have queues to store packet flits. The router arbiter grants an output port to a given input flow. To allow deriving WCET estimates for any task running in the system, no prioritization mechanism is used in the router, and arbitration decisions to select the flow accessing the requested output port are taken using a time-analyzable arbitration policy, e.g. round-robin.

We consider a *NxM* mesh NoC in which each node can be identified using (x,y) coordinates, see Figure 1(a). The router located at coordinates (x,y) is referred to as $R(x,y)$, see Table I for the definitions used in this paper. Each node comprises the router that communicates the node to the mesh and a *PME* (Processor/Memory element). The PME can be either a processor core, a cache memory, main memory, I/O, etc. In the network several traffic flows may exist. A traffic-flow (F_i) is a packet stream that traverses the same *H*-node route from a source to a destination node and requires the same grade of service along the path.

For the characterization we use deterministic *XY* routing but any other deterministic routing can be used too. Deterministic routing allows identifying routers in a given path as R^j where j is the hop number of the path (e.g. R^1 is the source node). With XY routing, packets are forced to use the X dimension first. In the X dimension the position of the target node with respect to the source node determines whether to go right (X+)

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

or left (X-) direction. The same approach is used for the Y dimension. Once packets are routed using the Y dimension they cannot be forwarded to the X dimension. Note that the opposite port is represented as $\bar{Y}$ and $\bar{X}$. For instance, the opposite port of $Y+$ is $Y-$. Routing restrictions help determining the exact number of requests (P_i^j) that might contend at router R^j for the same output port as F_i in the worst-case situation. For instance, P_i^j values for a mesh with XY routing and assuming all-to-all communication are determined as follows:

$$P_i^j = \begin{cases} 2 & if\ destination\ is\ X+\ or\ X- \\ 4 & if\ destination\ is\ Y+,\ Y-\ or\ PME \end{cases}$$

B. Deriving Worst-Contention Delay

In this section, we provide expressions to compute worst-contention delay (WCD) bounds that are also suitable for NoCs using weighted round-robin arbitration. Expressions given in this section are based on the concept of worst-case ejection rate (ER_i^j). We define ER_i^j as the rate at which flits of flow F_i can be ejected from router R^j to the corresponding port when the next router (R^{j+1}) is accepting incoming packets (i.e. it is not stalling R^j packet transmission). We also extend the concept of worst-case network ejection rate to model the rate at which flits can be ejected from a given router port when the network is fully congested. To do so, we define propagated worst-case ejection rate PER^{wc} as the minimum rate at which flits of F_i can be ejected from R^j in the worst-case situation. ER_i^j values can be computed by considering the maximum number of flows P_i^j contending at R_i^j for the same output port as F_i, see Equation 1.

$$ER_i^j = \frac{1}{P_i^j} \tag{1}$$

$PER_i(R^j)$ is computed by multiplying ER_i^j values from the current router R_i^j to the target router R_i^H as presented next:

$$PER_i^j = \prod_{k=j}^{H} \frac{1}{P_i^k} \tag{2}$$

Let D_i^j be the time that a packet of flow F_i requires to go from the input port of R^j to its target node. D_i^j can be computed recursively by considering the time required to reach R^{j+1} as $1/PER_{fx\{i\}}^j$ plus the time required to reach its destination once at R^{j+1}. $fx\{i\}$ represents the index of the flow that causes the worst possible blocking in F_i. Note that a $F_{fx\{i\}}$ packet stalled in a subsequent router of the path followed by F_i might cause F_i to suffer worst contention than one following exactly the same path. In the same way, $PER_{fx\{i\}}^j$ represents the worst ejection rate for F_i packets. To determine the flow causing the worst contention, PER values for all routers and all flows are computed in advance, and for any particular flow and router we choose the worst $PER_{fx\{i\}}^j$. Equation 3 shows the recursive definition of D_i^j.

TABLE II: WCD values for L-flit packets

	Round-Robin				Weighted			
	F_1	F_2	F_3	F_4	F_1	F_2	F_3	F_4
D_i^3	3L	-	-	-	2L	-	-	-
D_i^2	9L	3L	3L	-	6L	2L	4L	-
$D_i^1(WCD)$	15L	9L	6L	3L	10L	6L	8L	4L

$$D_i^j = \frac{1}{PER_{fx\{i\}}^j} + D_i^{j+1} \tag{3}$$

The WCD for flow F_i, given by D_i^1, is the time required to reach its destination ($j = H$) from the source node.

C. Computing WCD with Weighted arbitration

We illustrate how to compute WCD using equations above with the example presented in Figure 1(b) and considering round-robin arbitration first. We aim at computing F_1 WCD, i.e. the WCD of packets with source node in (x,y) router and destination in (x+1,y+1). First, we compute PER_i^j as the product of the ER_i^j coefficients (shown in brackets in Figure 1(b)) of all the routers that F_i ($i = 1$) traverses. Later, we start from the last hop ($j = 3$) and compute all D_i^j values;

$$D_1^3 = \tfrac{1}{1/3} = 3 \quad D_1^2 = \tfrac{1}{1/6} + D_1^3 = 9$$
$$D_1^1 = \tfrac{1}{1/6} + D_1^2 = 15$$

Figure 1(b) shows WCD values for the 2x2 NoC for both round-robin and weighted round-robin arbitration. In particular, we compute the WCD of F_1, F_2, F_3 and F_4. WCD for F_3 is given by $D_3^1 = D(x, y+1)$. Table II shows the D_i^j and WCD values for F_1, F_2, F_3 and F_4 for both round-robin and weighted arbitrations. F_4 comes directly from one of the router (x+1,y+1) ports. As shown in the table, weighted arbitration makes the WCD of the packets to be reduced for those flows that are located in the farthest positions, and this comes at the expense of penalizing the WCD of the nodes closer to the destination. It is also important to mention that despite weighted arbitration has the potential to fairly share the bandwidth, the WCD of each core is not fully equalized.

D. Limitations of Weighted Meshes

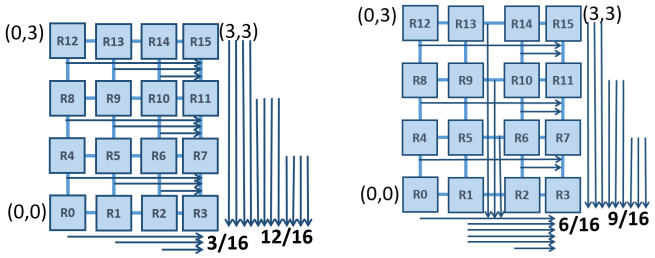
Weighted meshes have been proposed jointly with XY routing[1]. Figure 2(a) shows an example of XY routing in which for a typical 4x4 mesh with the target node (e.g. a shared cache or main memory) connected to a port in a router in one corner (e.g. (3,0)) and all nodes attempting to send packets to that node.

As shown, under XY routing the router at the corner has to arbitrate among three input ports with highly unbalanced traffic: (1) the X+ port receives requests from 3 out of 16 cores, (2) the local port (not shown in the picture) receives requests from 1 out of 16 cores, and (3) the Y+ port receives requests from the remaining 12 cores. Overall, the Y+ port needs to absorb 75% of the traffic if bandwidth across cores is homogeneously balanced.

[1]Note that the reasoning that follows applies identically with YX routing.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a) XY mesh (b) XY YX mesh

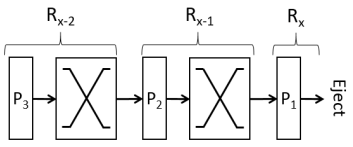
Fig. 2: Routes to router (3,0).

Let us consider an arbitration window implemented as an extended round-robin vector with 16 entries, so that bandwidth can be weighted as needed for homogeneous bandwidth allocation. The arbitration window for the router in the corner is composed of 16 elements (1 PME, 3 X+, 12 Y+). The only way to arrange those port grants in a window so that, inside the window and across windows, the maximum number of consecutive arbitrations granted to the same port is minimized is using the following pattern: $PME, Y+, Y+, Y+, X+, Y+, Y+, Y+, X+, Y+, Y+, Y+, X+, Y+, Y+, Y+$. Note that swapping PME and $X+$ grants or moving $Y+$ grants to the beginning would lead to equivalent patterns with $Y+, Y+, Y+$ sequences interleaved with other (individual) grants over time.

The effectiveness of the weighted arbitration is decreased in the presence of bubbles. In regular wormhole mesh NoC designs, bubbles can occur due to local and global control-flow effects.

Canonical routers in a 2D mesh are pipelined into several stages. First, the incoming packet is stored in the corresponding input buffer. Then, routing and switching occur in one ore more stages, and finally the packet is sent through the link. However, since no packet-loss is allowed in wormhole, before the packet is sent to the next router, the stall/go signal coming from the next router is checked to ensure there is enough buffer space to store the packet. This stall/go signal is used to ensure the link-level (or local) flow control and requires C_f cycles to travel from the destination router (R_{x+1}) to the current one (R_x), and the round trip time (RTT) is equal to $2 \times C_f$. The RTT determines the amount of buffering required at the input buffers to avoid having bubbles in the transmission.

Let us consider the example in Figure 3, where we show the main stages of three consecutive routers (R_x, R_{x-1}, R_{x-2}), being packets ejected through router R_x. Let us also consider three packets (P_1, P_2, P_3), and an initial state where P_1 is in an input buffer in R_x, P_2 in an input buffer of R_{x-1}, and P_3 in an input buffer of R_{x-2}. If R_x did not allow these packets to make any progress in the previous cycle, but R_{x-1} and R_{x-2} allowed them to progress, P_1 could not be switched and ejected. Instead, P_2 and P_3 where switched but could not be transmitted to the following router due to backpressure of the input buffers. This is illustrated in the chronogram in Figure 3 as the state in cycle 0. Eventually, in cycle 1 P_1 is ejected, P_2 reaches the input buffer of R_x, and P_3 the input buffer of R_{x-1}. In cycle 2 P_2 is ejected, and P_3 is switched in R_{x-1}. In cycle 3, P_3 reaches the input buffer of



	0	1	2	3	4
P_1	B(x)	Eject			
P_2	sw(x-1)	B(x)	Eject		
P_3	sw(x-2)	B(x-1)	sw(x-1)	B(x)	Eject

Fig. 3: Example of bubbles when sending continuously packets from one port.

R_x, but cannot be ejected until cycle 4. Overall, R_x can eject up to 2 packets consecutively coming from the same input port. Hence, a weighted mesh where more than 2 packets need to be transmitted in consecutive cycles cannot serve those packets at a sufficient speed and causes some imbalance.

In this example, if buffers are large enough potentially all packets can reach R_x faster. However, buffers are the most expensive resource in routers, so their size is kept as low as reasonably possible.

Global Effects. In current CRTES, packets in the NoC usually correspond to memory transactions going from the cores to the memory devices and the other way around. The maximum speed at which cores issue requests to memory is determined by the amount of cycles the requests going to memory need to be processed. In the context of a NoCs, this time is usually in the order tens of cycles and avoids having always requests to be served by the router during operation. However, WCD estimation cannot make any assumption on the actual load, so worst congestion must be assumed for WCET estimation.

Thus, even allocating bandwidth as in [15] (that theoretically allows achieving homogeneous arbitration using weights that consider the amount of flows using each input port), global fairness cannot be achieved in practice if the amount of traffic flows traversing each port is not balanced as well.

IV. EOMESH: A Fair Weighted Mesh

As we have seen in previous section, although regular weighted NoCs with XY routing are theoretically able to achieve a perfect balancing of the available bandwidth, this balancing is not achieved in reality due to the presence of bubbles. To solve this problem, we propose a new mesh design intended to achieve near-optimal weighted arbitration. The idea behind Even/Odd mesh (EOmesh) is that the even allocation of bandwidth cannot be practically achieved by simply playing with the arbitration but it also requires balancing the amount of flows served at each output port. EOmesh combines both concepts and it balances both (1) the bandwidth each flow is assigned using a weighted arbitration and (2) the amount of flows each port has to serve. In the following sections we describe how EOmesh implements these concepts.

A. Combined Flow Balancing & XY-YX routing

In order to balance the amount of flows each output port serves, we change the routing algorithm. With XY routing, the default routing policy in the weighted mesh proposed in [17], ports in the Y direction serve high number of flows that makes not possible to achieve a fair distribution of the bandwidth due to the presence of bubbles. To avoid the high concentration of

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

flows in a given port we propose a different routing algorithm that combines both XY and YX to achieve a better balancing of the flows traversing each port. The proposed algorithm uses XY for packets originated at the source nodes with an even identifier, and YX for the packets from source nodes with an odd identifier. The proposed mechanism is illustrated in Figure 2(b): combining XY and YX algorithms for even and odd sources allows improving the balancing in the number of flows each port has to serve. For the particular example in the plot, we see how the output port arbitration of the memory controller attached to node 3 serves 6, 9, and 1 flows through the X, Y, and PME input ports while in the case of the XY routing (Figure 2) the distribution of flows is 3, 12, 1, for the X, Y, and PME input ports.

Virtual channel allocation. The main reasons why XY routing is heavily utilized are its simplicity (low complexity) and its deadlock freedom properties. However, as we have analyzed previously, XY routing does not balance traffic flows efficiently. With EOmesh routing approach, the amount of flows traversing each port is more balanced but on the other side combining XY and YX allows creating deadlock situations for specific communication flows. In that respect, in order to avoid deadlock situations, EOmesh assigns a specific virtual channel to each of the two routing policies employed (XY and YX). Performing a static virtual channel allocation to isolate flows from the different routing policies XY and YX also allows reducing the worst contention that the different flows can have since the amount of contender packets that each flow finds in each hop is reduced.

Given that dynamic virtual channel allocation policies have a very negative impact in WCD – since the number of potential contenders is increased in a router with virtual channels [16] – the static allocation of VCs performed by EOmesh does not degrade the best guaranteed performance achievable by the network.

B. Adapting arbitration weights

Weights in the weighted round-robin mesh design proposed in [17] can be computed using the following expression:

$$w(I_{dir}, O_{dir}) = I_{dir}/O_{dir} \quad (4)$$

where I_{dir} represents the number of communication flows traversing the dir_i input port of a given router being dir any of the possible mesh router port directions. Similarly, O_{dir} is the number of flows traversing the dir output port of the same router.

Given a fixed number of communication flows, the actual flows traversing the input/output ports of each router can be determined considering the particular route used by each flow. Figure 4 shows the EOmesh weights required to arbitrate the flows originated at each router that target the shared resource attached to Router (3,0)).

V. Implementation

The baseline weighted mesh can be implemented in two main different ways: (1) as a programmable NoC or (2) as a hardwired NoC.

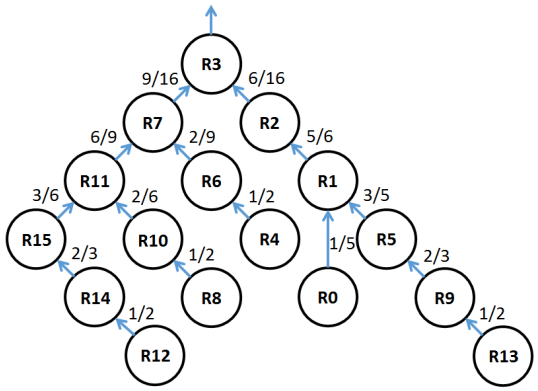


Fig. 4: EOmesh arbitration weights to access a shared resource attached to router (3,0). Router internal port (PME) weights are not shown in the picture.

A. Programmable NoCs

In a programmable NoC, routing and arbitration decisions can be interfaced and modified by means of software commands, thus with no hardware modification.

Programmable routing. Making routing programmable requires, for instance, the use of routing tables in each port. Hence, for each port of each router, we need a table with as many entries as potential different flows (e.g. 16 entries for a 4x4 mesh) where each entry contains the identifier of a destination port, which can be $\{X+, X-, Y+, Y-, PME\}$. Although 5 different values are possible across all ports, for a given port only 4 of them are possible since a port cannot be its own target. For instance, valid values for the $X+$ port are $\{X-, Y+, Y-, PME\}$. Thus, this table needs 2-bit entries to encode the destination port for each flow in each input port. Overall, given a NxM mesh, the (distributed) storage required to make routing programmable is:

$$RoutCost = (NxM) \cdot N_{ports} \cdot (NxM) \cdot bits/entry \quad (5)$$

Where the first NxM factor is the number of routers, N_{ports} stands for the number of ports per router (up to 5 ignoring the fact that routers at the boundaries do not have all ports), the second NxM factor corresponds to the number of entries per routing table, and the last factor, $bits/entry$ is 2 as indicated before. For instance, in a 4x4 mesh, routing tables require less than $16 \cdot 5 \cdot 16 \cdot 2 = 2560$ bits, so 320 bytes, which is a rather small cost. This design is sketched in Figure 5 (left).

Programmable arbitration. Two main alternative implementations can be used to have programmable arbitration. One of them requires an arbitration window per port and per router, with an arbitrary number of entries N_{arb}, and 2 bits per entry indicating the input port that is granted access to the particular output port. N_{arb} must be sufficiently large so that $1/N_{arb}$ provides sufficient granularity to allocate weights as needed. Moreover, N_{arb} may change across ports and routers. For the sake of illustration, we assume that arbitration windows have NxM entries. Then, each arbiter also needs a $log_2 N_{arb}$ counter pointing to the next entry in the window along with an incrementer for that counter. Alternatively, one could use shift registers with wrap-up for the window and use always

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

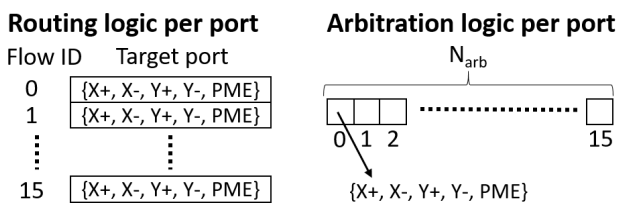


Fig. 5: Routing and arbitration implementation of a programmable NoC.

the value at a given position (e.g. first position) to determine what port is granted access next. In this work we build on the solution using the counter. Hence, the storage cost for a programmable arbitration is:

$$ArbCost = (NxM) \cdot N_{ports} \cdot ((NxM) \cdot bits/entry + log_2(NxM)) \quad (6)$$

For instance, in a 4x4 mesh, arbitration would require $16 \cdot 5 \cdot (16 \cdot 2 + 4) = 2880$ bits, so 360 bytes only. This design is sketched in Figure 5 (right).

A second alternative implementation of the arbitration can be built upon counters, where 4 counters are set per port, so that each one tracks how many times a given input port must be granted access to a given output port in each window. In general, this approach allows a finer-grain allocation of weights, but does not allow controlling with precision the order in which grants are given. For instance, it may grant access to the port with the highest count, or in a round-robin fashion across ports with non-zero counters.

VI. Evaluation

A. Hardwired NoCs

Some NoC implementations favor efficiency over flexibility, and routing and arbitration choices are hardwired. Adapting such a NoC to implement the EOmesh would require, at most, duplicating (simple) routing logic in some routers to implement XY and YX policies for different flows. In practice, since a packet can be forwarded in any direction depending on its destination at a given router, the practical cost of an additional check will have lower cost.

Regarding arbitration, using different weights for a weighted mesh would require hardwiring different choices in the arbitration windows, thus not increasing hardware cost.

Overall, hardware modifications would have limited impact on the overall cost of the NoC, which is mostly dominated by the buffering required at input ports.

B. Evaluation Framework

We evaluate EOmesh on a cycle-accurate simulator executing PowerPC ISA programs. The simulator is an enhanced version of SoCLib [28] that we integrated with gNoCsim [1], a cycle-accurate NoC simulator.

We evaluate EOmesh on meshes with 16 cores (4x4) and 36 cores (6x6) to assess the scalability of our approach.

In our manycore, load (and write-miss) requests comprise four-flit messages from the core to memory. Given that cache

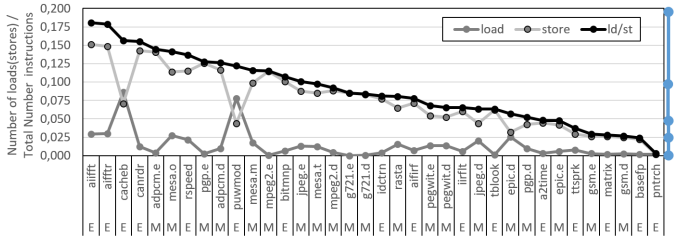


Fig. 6: Percentage of loads/stores in each benchmark. *EM* stands for EEMBC Auto and MediaBench.

line size is 64-bytes and we need 16-bits for control data (512+16 bits), memory answers with 4-flit messages over 132-bit wide links. Evicted line requests require a 4-flit message and a one-flit answer. In our proposed mesh design, packetization [16], shown to minimize WCD, adds control data to each of the flits, therefore requiring an extra flit, so 5 instead of 4 (512+5*16 bits over a 132-bit wide channel), leading to 25% overhead.

C. Metrics

For each setup, usually matching one result chart, WCET estimates are shown normalized to the maximum WCET observed in those results obtained with the baseline weighted mesh. Hence, normalized WCET (nWCET) estimates below 1 are better than those provided by the default weighted mesh and vice versa.

$$nWCET = \frac{WCET_{EOmesh}}{max(WCET_{weighted})} \quad (7)$$

D. Benchmarks and Workloads

Manycores can speed up the guaranteed execution time (i.e. reduce the WCET) of real-time multithreaded applications that use several cores. Manycores can also increase the number of single-task applications that can be safety consolidated. Interestingly, the former are becoming more frequent in automotive systems with the advent of autonomous driving, while the latter are more frequent in more conservative domains (e.g. avionics) that build on software partition-based systems (e.g. IMA [32] in avionics).

We aim at capturing the load that different applications put on the NoC. In order to use representative load values, we build on two representative suites: MediaBench [10] that comprises multimedia and communication applications relevant for autonomous navigation and driving systems, and EEMBC Autobench [18] benchmark suite that comprises automotive applications. Figure 6 shows the load each benchmark in both suites puts on the NoC: loads, stores and the addition of both. In particular we measure local data/instr. cache miss rate, breaking it down between loads and stores. We see variance in the load/store rate from 0% up to 15%.

We create different benchmarks (A, B, ..., H) with load and store access frequencies between 2.5% and 40%, with the following steps: 2.5%, 5%, 10%, 20%, and 40%, as shown on the right hand side of Figure 6. Hence we capture observed variance and also go beyond by mimicking more heavy loaded

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE III: Benchmarks

Benchmarks	A	B	C	D	E	F	G	H
% Local Op	80	50	50	60	95	87.5	87.5	90
% LD Op	10	10	40	20	2.5	2.5	10	5
% ST Op	10	40	10	20	2.5	10	2.5	5

scenarios. Table III shows the details on our benchmarks from which we generate several workloads to evaluate EOmesh.

- Analytical maximum load. WCD bounds are computed for all cores. This is the most stressful scenario for the NoC, where requests are assumed to contend in the worst possible way with other cores' requests.
- Independent applications. We generate a set of applications with varying number of load (LD), store (ST) and in-core (local) operations as described in Table III. Those allow assessing different degrees of load in the NoC. In particular, benchmark A-D correspond to high-demanding benchmark in terms of bandwidth, whereas E-H benchmark are their low-demanding counterparts (25% requests w.r.t. A-D applications).
- Parallel applications. We create 2x2 homogeneous parallel applications in which all threads correspond to the same reference benchmark to study their sensitivity to the particular mapping of the application in the mesh.
- Heterogeneous applications. We have generated heterogeneous parallel applications comprising several phases and varying the amount of tasks in each of the phases. We have generated 8000 directed acyclic graphs (DAG) to simulate different coarse-grain parallel applications as the ones supported by the ADA programming language [23]. Figure 7 shows an schematic of the DAG template we have used to generate the different applications. For each application DAG tasks are chosen randomly from the workloads in Table III.

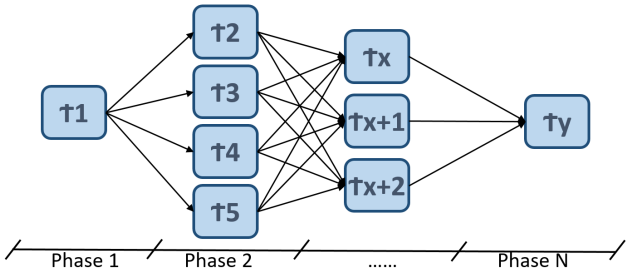


Fig. 7: DAG schematic of the heterogeneous applications.

E. Analytical Contention Bounds

First, we compute the WCD bounds across cores, whose maximum determines the worst contention that any core request could experience. As shown in Figure 8 for a 4x4 mesh, the EOmesh reduces significantly the WCD for those flows (the corresponding router is indicated in brackets) with highest WCD values. This is achieved by a better organization of the traffic (combined XY and YX routing policy) and an appropriate weight allocation, which slightly increases the WCD for the fastest nodes to decrease it for the slowest ones.

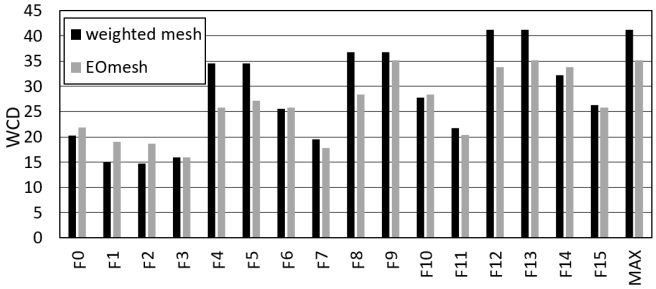


Fig. 8: WCD bounds for all flows (4x4 mesh nodes).

In particular, the maximum WCD for the original weighted mesh (flows 12 and 13) decreases by 14.7% in the EOmesh (flow 13). Thus, the amount of contention that is added to each request during WCET estimation is significantly reduced.

EOMesh is specifically designed to reduce the WCET of the slowest thread in a parallel application. This is achieved by allocating bandwidth so that the maximum WCD (and so the imbalance) in the original wmesh is reduced. That is, the WCD (and hence the WCET) suffered by threads closer to main memory is on-purpose increased to reduce that of threads farther away from main memory, since those determine the WCET of the overall application. In this line, Figure 8 should not be read in terms of the number of threads (nodes) for which EOMesh reduces WCD, but instead, on whether EOMesh reduces the WCET of the slowest thread.

F. Independent (single-threaded) Applications

In order to assess the impact of EOmesh WCD reduction w.r.t. the baseline weighted mesh, we have evaluated it using all single-threaded applications. In particular, we obtain time-composable WCET bounds (thus valid regardless of the tasks running in the other cores) for each application on each of the cores. WCET estimates have been obtained by considering the execution time in isolation (obtained with measurements in this particular evaluation) and adding the WCD to each request, where the WCD depends on the mesh policies (baseline weighted mesh or EOmesh) and the particular core where the application is run. Note that other WCET estimation practices applied during unit testing (aka when contender tasks are unknown) would provide different in-isolation execution times, but would account for NoC contention analogously to our work (i.e. considering the worst-case contention per request).

Figure 9 shows, for each application, the WCET reduction achieved with the EOmesh $(1 - \frac{WCET_{EOmesh}}{WCET_{Weighted}})$. As shown, despite the fraction of operations causing NoC requests decreases from 50% (applications B and C) down to only 5% (application E), the relative WCET reduction in the range 5%-9% and 15%-28% for 4x4 and 6x6 meshes respectively due to the fact that NoC latency is the dominant factor in the WCET time. For completeness, we also show the results in a per-core basis for application A (see Figures 11 and 12 for 4x4 and 6x6 respectively), as illustrative example of how the WCET varies across cores. Notice that in this case we use the nWCET metric in Equation 7. As shown, in line with WCD results, the EOmesh slightly penalizes the WCET for those cores with lowest in-isolation execution time to decrease appreciably the WCET in the cores with highest in-isolation execution time.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

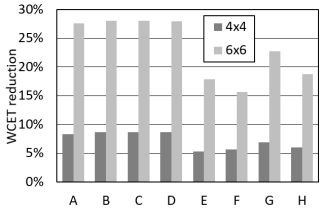


Fig. 9: WCET estimation reduction.

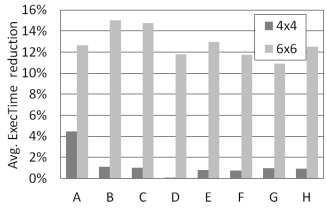


Fig. 10: Average execution time reduction.

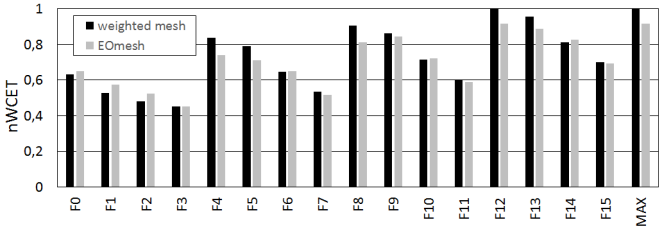


Fig. 11: Normalized WCET for application A (4x4).

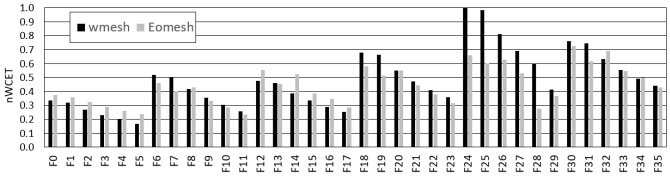


Fig. 12: Normalized WCET for application A (6x6).

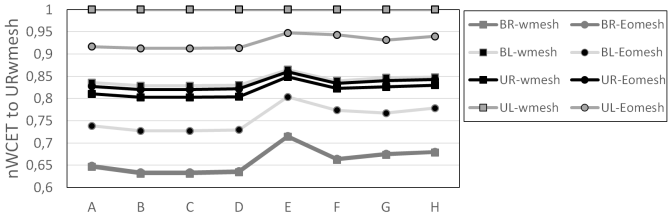


Fig. 13: nWCET. 4-thread homogeneous apps (4x4 mesh).

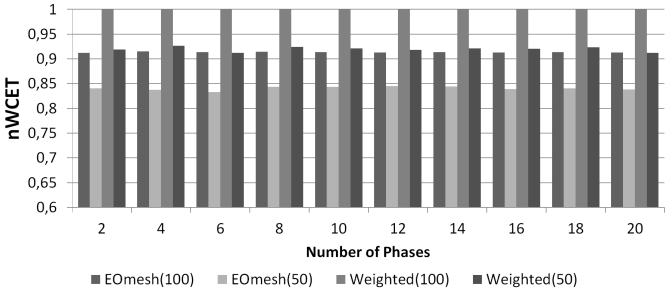


Fig. 14: nWCET for heterogeneous parallel applications.

G. Parallel Applications

We have evaluated the relative WCET of a set of parallel applications building on our reference single-threaded applications on a 4x4 mesh. The first experiment consisted in building 16-thread parallel applications with all threads being of the same type (e.g. 16 A applications). As expected, the WCET was dictated by the worst thread, thus delivering the same results shown in Figure 9. Then, we have built 4-thread parallel applications mapping them in 4 different square regions (2x2 cores) in the 4x4 mesh. Each such square region corresponds to the partition of the mesh into 4 square regions, which we identify with U (upper), B (bottom), L (left) and R (right). For instance, UR corresponds to the upper-right square, which includes nodes (2,2), (2,3), (3,2) and (3,3).

Results for those 4-thread homogeneous parallel applications are shown in Figure 13, again, normalized (for each reference application) w.r.t. the maximum WCET across all parallel applications. As shown, the EOmesh provides significant gains for the UL mapping, which is the one with highest WCET estimates. Gains for the second worst case (BL) are also noticeable. Results for the third worst case (UR) are slightly worse for the EOmesh. As explained before, the EOmesh decreases the WCD of the slowest nodes at the expense of increasing the WCD for some (fast) nodes. Finally, the best square (BR) experiences almost no change (up to 0.3% WCET variation).

Overall, the EOmesh proves to be also beneficial for parallel applications, decreasing the WCET in those regions that lead to highest WCET estimates.

Figure 14 shows results for DAG-based heterogeneous parallel applications on a 4x4 mesh, whose threads are mapped randomly to cores. We show results for applications with 1 to 20 phases and processor utilizations of 50% and 100%. A 100% utilization means that all the 16 cores in the manycore are utilized while with 50% utilization the amount of cores utilized in each of the phases is different but being 50% on average. As show in the plot, WCET reductions around 8%-9% are obtained regardless the degree of utilization and the number of phases of the parallel applications. EOmesh effectively homogenizes bandwidth across cores, which ultimately decreases the imbalance across threads in parallel applications. While some threads run in cores with lower bandwidth in the EOmesh, since some others run in cores with low bandwidth in the baseline weighted mesh allows the EOmesh to speed them up. Hence, by speeding up the threads in the slowest cores (due to their lower bandwidth), EOmesh improves the performance of parallel applications.

H. Average Performance

One of the main side effects when balancing the behavior of applications – and quality of service solutions in general – is the loss of throughput, since processor design is tunned to improve average performance. In order to show that our proposed approach does not negatively impact average performance, we have run our reference applications in all cores simultaneously and collected measurements with actual contention. Our results show that, in general, the discrepancies between the baseline weighted mesh and the EOmesh are very low. For illustration purposes, in Figure 10 we show the execution time reduction obtained with the EOmesh in the slowest core in each case. As shown, EOmesh does not cause any degradation in performance but results in some gains (around 1%-2% for 4x4 and 11%-15% for 6x6) due to avoiding

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

bubbles in arbitration, thus removing unnecessary stalls and increasing also average performance.

VII. Related Work

While there have been several proposal for real-time aware NoC designs, exploring to which extent high-performance (COTS) NoC designs can be used in the real-time domain is of paramount importance: on the one hand, it is well accepted that the CRTES domain is a relative small market in comparison with other domains such as mobile. Hence, customized NoCs specifically designed for real-time systems (e.g. time-triggered ones and those based on TDMA), which may require high non-recurrent costs, will naturally find difficulties to be adopted in the context of industrial CRTES [29]. On the other hand, the big majority of the proposed manycore designs across all computing domains use high-performance wormhole NoCs (wNoCs) to perform the interconnection of cores and shared resources within the chip. This makes wNoCs accessible (at low cost) by the CRTES since they are implemented in a vast set of chips. In this paper, we have focused on achieving global balanced bandwidth as a way to reduce WCET estimates with no (or minimum) hardware support.

Several real-time specific NoCs have been proposed based on TDMA such as [25] and [5]. While TDMA-based NoCs deal with contention at transaction level (e.g. read and write memory operations), time-triggered architectures [13] increase the abstraction level by introducing a self-contained computational unit. In time-triggered architectures, micro-components exchange messages in contention-free slots. However, event-triggered transactions, such as cache misses that access main memory through the NoC, may suffer contention delay, which must be upper bounded. We refer to NoC designs with real-time guarantees and time-composable behavior as guaranteed service NoCs. Nostrum [12] and Aethereal [5] NoCs provide guaranteed service using time-division multiplexing, and hence, time composable bounds.

Many studies have also been carried out with the purpose of providing realistic and feasible latency bounds for best-effort wNoCs. Using prioritization on a per-virtual channel basis has proven being an effective means to achieve tight latency bounds in wNoCs [26]. However, the use of per-virtual channel prioritization becomes impractical when a significant number of flows exist in the network. To overcome this issue, the impact of virtual channel sharing has been analyzed in [27] and [21]. However, while these approaches effectively reduce the number of virtual channels required, the timing guarantees obtained build upon a detailed knowledge of the characteristics of the software (applications and/or tasks) that will be executed in the deployed system and hence, do not meet incremental qualification requirements. The work in [8] has similar pros and cons, since the proposed solution guarantees specific bandwidth allocation for guaranteed-service connections per port, by splitting the bandwidth of output ports among best effort and guaranteed service connections.

Authors in [9] made one of the first studies that provided reliable contention bounds for wNoCs without building upon flit-level virtual channel preemption. Later, this analysis has been improved in [20] where tighter bounds are presented. The model in [20], as those mentioned above, also requires detailed information on all communication flows that will be finally deployed in the system to estimate reliable upperbounds. In other words, latency bounds provided in [20] are not time-composable. Some recent works that build upon wNoCs propose interference-free NoC designs [19], [31]. The solution in [19] has been proven to cause lower degradation on best-effort traffic than the one in [31]. The former achieves its goal by using specific ways to multiplex virtual channels. However, despite its improved performance, the performance degradation caused on best-effort traffic is still large.

Recently, authors in [15] have proposed an alternative approach to meet CRTES requirements. In particular, that work proposes specific ways to derive time-composable WCD bounds without sacrificing average performance and by allocating weights to arbiters so that fair bandwidth allocation is achieved across cores, building upon weighted meshes, which are widely used in high-performance routers for off-chip wormhole networks [3] . Still, as discussed before, bandwidth allocation is improved but is still unbalanced. In our work we tackle this limitation by adapting routing policies and weight allocation conveniently.

VIII. Conclusions

Weighted meshes are an effective solution to homogenize bandwidth allocation across cores, which is of prominent importance for tight WCET estimation in critical real-time systems. However, as shown in this work, weighted meshes fail to provide homogeneous bandwidth across cores. In this work we introduce the EOmesh that provides near-optimal homogeneous bandwidth allocation across cores by (i) combining XY and YX routing policies and (ii) allocating weights accordingly. This allows limiting the local bandwidth allocated to specific ports that cannot use all allocated slots, thus inducing some imbalance. Our results show large WCD reductions and WCET reductions in the range 5%-28%, while keeping hardware cost roughly unchanged. Moreover, we show that benefits hold in the context of parallel applications, whose WCET becomes less sensitive to the particular cores where they are allocated.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Carles Hernández is jointly funded by the MINECO and FEDER funds through grant TIN2014-60404-JIN. Francisco J. Cazorla is partially funded by the European Research Council (ERC) under EU's H2020 research/innovation programme (grant No. 772773)

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2857298, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

REFERENCES

[1] *NanoC: NaNoC design platform. http://www.nanoc-project.eu*.

[2] ARM. ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade, 2015. https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decad php.

[3] D. Crupnicoff et al. Deploying quality of service and congestion control in infiniband-based data center networks. *White paper, Mellanox Technologies*, 2005.

[4] GENESYS. GENeric Embedded SYStem Platform, 2010. http://www.genesys-platform.eu.

[5] K. Goossens, et al. Aethereal network on chip: concepts, architectures, and implementations. *Design Test of Computers, IEEE*, 2005.

[6] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[7] J. Jalle et al. Deconstructing bus access control policies for real-time multicores. In *SIES*, June 2013.

[8] T. Kranich and M. Berekovic. Noc switch with credit based guaranteed service support qualified for GALS systems. In *DSD*, 2010.

[9] S. Lee. Real-time wormhole channels. *Journal Of Parallel And Distributed Computing*, 63:299–311, 2003.

[10] C. Lee et al. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 30, pages 330–335, 1997.

[11] MERASA. *EU-FP7 Project: www.merasa.org*, 2010.

[12] M. Millberg et al. The nostrum backbone-a communication protocol stack for networks on chip. In *IEEE VLSI Design*, pages 693–696, 2004.

[13] R. Obermaisser et al. The time-triggered system-on-a-chip architecture. In *ISIE*, 2008.

[14] M. Panic et al. Parallel many-core avionics systems. In *EMSOFT*, 2014.

[15] M. Panic et al. Improving performance guarantees in wormhole mesh noc designs. In *DATE*, 2016.

[16] M. Panic et al. Modeling high-performance wormhole nocs for critical real-time embeddedd systems. 2016.

[17] H. Park and K. Choi. Position-based weighted round-robin arbitration for equality of service in many-core network-on-chips. NoCArc, 2012.

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

[19] A. Psarras et al. Phase-noc: Tdm scheduling at the virtual-channel level for efficient network traffic isolation. In *DATE*, 2015.

[20] D. Rahmati et al. Computing accurate performance bounds for best effort networks-on-chip. *IEEE Transactions on Computers*, 2013.

[21] E. A. Rambo and R. Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. DATE, 2015.

[22] J. Rattner. *Single-chip Cloud Computer: An experimental many-core processor from Intel Labs*, 2015.

[23] S. Royuela et al. Openmp tasking model for ada: Safety and correctness. In *Reliable Software Technologies - Ada-Europe*, pages 184–200, 2017.

[24] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[25] M. Schoeberl et al. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *IEEE/ACM NoCS*, 2012.

[26] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *NoCS*, 2008.

[27] Z. Shi and A. Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *ECRTS*, 2009.

[28] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.

[29] J. Sparsoe. Design of networks-on-chip for real-time multi-processor systems-on-chip. In *ACSD*, 2012.

[30] Tilera. *TILE-Gx Processors Family http://www.tilera.com/products/TILE-Gx.php*.

[31] H. M. G. Wassel et al. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. *SIGARCH Comput. Archit. News*, 41(3):583–594, June 2013.

[32] C. Watkins and R. Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. *In Proceedings of 26th Digital Avionics Systems Conference. DASC '07*, 2007.

[33] R. Wilhelm et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.

AUTHORS

Jordi Cardona is a PhD. Student for the CAOS group at BSC. He obtained his M.S. degree in 2018 and graduated in Informatics Engineering in 2016, both titles obtained from the Universitat Politècnica de Catalunya. He enrolled BSC in 2016 where he started working on the analysis of COTS networks on chip for real-time multi-core systems during his master thesis and his current research focuses on monitor and control contention in shared resources of critical real-time systems.

Carles Hernández received the M.S. degree in telecommunications and PhD in computer sciences from Universitat Politècnica de València, in 2006 and 2012, respectively. He is currently senior PhD. Researcher at the Barcelona Supercomputing Center. His area of expertise includes network-on chip and reliability-aware processor design. He participates (has participated) in NaNoC, parMERASA, PROXIMA IP7, VeTeSS ARTEMIS, and RECIPE H2020 projects. In the context of RECIPE he is leading the part on timing analysis of high-peformance hardware designs. In 2012 he was intern at Intel Mobile Communications Munich.

Jaume Abella is a senior PhD. Researcher in the CAOS group at BSC and member of HIPEAC. He received his MS (2002) and PhD. (2005) degrees from the UPC. He worked at the Intel Barcelona Research Center (2005-2009) in the design and modeling of circuits and microarchitectures for fault-tolerance and low power. He joined the BSC in 2009 where he has been in charge of hardware designs for FP7 PROARTIS and PROXIMA, and been the BSC PI for ARTEMIS VeTeSS and H2020 SAFURE. Jaume is (and has been) also involved in several ESA-BSC bilateral projects and other bilateral industrial projects with avionics and automotive industry. He has authored 15 patents and more than 100 papers in top conferences and journals in the area. He is (has been) co-advisor of 15 MS and PhD students.

Francisco J. Cazorla is the leader of the CAOS group at BSC and member of HIPEAC Network of Excellence. He has led projects funded by industry (IBM and Sun Microsystems), by the European Space Agency (ESA) and public-funded projects (FP7 PROARTIS project and FP7 PROXIMA project). He has participated in FP6 (SARC) and FP7 Projects (MERASA, VeTeSS, parMERASA). His research area focuses on multithreaded for both high-performance and real-time systems on which he is co-advising several PhD theses. He has co-authored 3 patents and over 100 papers in international refereed conferences and journals. He is an ERC consolidator grant holder.

0278-0070 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.