On-Chip Interconnects and Instruction Steering Schemes for Clustered Microarchitectures

Joan-Manuel Parcerisa[1], Julio Sahuquillo[2], Antonio González[1,3], José Duato[2]

[1] *Dept. Arquitectura de Computadors*
Universitat Politècnica de Catalunya
Barcelona, Spain
{jmanel,antonio}@ac.upc.es

[2] *Dept. Informàtica de Sistemes i Computadors*
Universitat Politècnica de València
València, Spain
{jsahuqui,jduato}@disca.upv.es

[3] *Intel Barcelona Research Center*
Intel Labs
Univ. Politècnica de Catalunya
Barcelona, Spain

Abstract

Clustering is an effective microarchitectural technique for reducing the impact of wire delays, the complexity, and the power requirements of microprocessors. In this work, we investigate the design of on-chip interconnection networks for clustered superscalar microarchitectures. This new class of interconnects has demands and characteristics different from traditional multiprocessor networks. In particular, in a clustered microarchitecture, a low inter-cluster communication latency is essential for high performance.

We propose some point-to-point cluster interconnects and new improved instruction steering schemes. The results show that these point-to-point interconnects achieve much better performance than bus-based ones, and that the connectivity of the network together with effective steering schemes, are key for high performance. We also show that these interconnects can be built with simple hardware and achieve a performance close to that of an idealized contention-free model.

Keywords: Clustered microarchitecture, inter-cluster communication, on-chip interconnects, instruction steering, complexity

1. Introduction

Superscalar architectures have evolved towards higher issue-widths and longer instruction windows in order to achieve higher instruction throughput by taking advantage of the ever increasing availability of on-chip transistors. These trends are likely to continue with next generation multithreaded microprocessors [16, 34], which allow for a much better utilization of the resources in a wide issue superscalar core.

However, increasing the complexity also increases the delay of some architectural components that are in the critical path of the cycle time, which may significantly impact performance by reducing the clock speed or introducing pipeline bubbles [21]. On the other hand, projections about future technology trends foresee that long wire delays will scale much slower than gate delays [1, 6, 14, 17, 19]. Consequently, the delay of long wires will gradually become more important.

Clustering of computational elements is becoming widely recognized as an effective method for overcoming some of the scaling, complexity, and power problems [4, 11, 12, 13, 21, 24, 32, 34, 38]. In a clustered superscalar microarchitecture, some of the critical components are partitioned into simpler structures and are organized in smaller processing units called clusters. In other words, a clustered microarchitecture trades-off IPC for a better clock speed, energy consumption, and ease of scaling.

While intra-cluster signals are still propagated through fast interconnects, inter-cluster communications use long wires, and thus, are slow. The impact of these communication delays is reduced as far as signals are kept local within clusters. Previous work showed that the performance of a clustered superscalar architecture is highly sensitive to the latency of the inter-cluster communication network [8, 24]. Many steering heuristics have been studied to reduce the required communications [5, 9, 22], and value prediction has been proposed to hide the communication latency [24]. An orthogonal approach proposed in this paper consists of reducing the communication latency by designing networks that reduce the contention delays and proposing effective improvements to the instruction steering scheme that minimize both the communication rate and the communication distance. Moreover, the proposed interconnects also reduce capacitance, thus speeding up signal propagation.

For a 2-cluster architecture it may be feasible to implement an efficient and contention-free cluster interconnect by directly connecting each functional unit output to a register file write port in the other cluster. However, as the number of clusters increases, the fully connected network may be very costly or unfeasible due to its complexity. On the other hand, a simple shared bus requires lower complexity but it has high contention. Therefore, a particular design needs to trade-off complexity for latency to find the optimal configuration.

Previous works on clustered superscalar microarchitectures have assumed interconnection networks that are either an idealized model ignoring complexity issues [5, 24], or they consider only 2 clusters (Multicluster [11], Alpha 21264 [13]), or they assume a simple but long-latency ring [2,

3, 15]. In this paper, we explore several alternative interconnection networks with the goal of minimizing latency while keeping the cluster complexity low. We have studied two different technology scenarios: one with four 2-way issue clusters, the other with eight 2-way issue clusters. In both cases, we propose different point-to-point network topologies that can be implemented with low complexity and achieve performance close to that of idealized models without contention. This paper extends the work in [26] in several ways; i.e., we propose a new steering algorithm, we run simulations using two benchmark suites, and we analyze more deeply the router operation and design.

The rest of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 gives an overview of the assumed clustered microarchitecture. Section 4 describes the proposed cluster assignment schemes. Section 5 discusses the main design issues of the proposed interconnects. The analyzed interconnect models for four and eight clusters are described in sections 6 and 7, respectively. Section 8 analyzes the experimental results, and finally section 9 summarizes the main conclusions of this work.

2. Related Work

Aggarwal and Franklin evaluated a crossbar and a ring for a clustered superscalar architecture with a centralized register file (the PEWs processor) [2] and later they extended their study with a hierarchical ring of crossbars for a large number (8 to 12) of 2-way issue clusters [3]. The topology they propose connects a small number of physically close clusters using a low-latency crossbar, while the ring connects distant clusters. Their approach mainly differs from ours because it focuses more on the scalability of the steering algorithms than on the design of the interconnects themselves.

More recently, Sankaralingam *et al.* [30] describe a taxonomy of inter-ALU networks which includes, among others, conventional broadcast schemes as well as multi-hop interconnects. Through detailed circuit analysis, they estimate communication delays for single-hop and multi-hop interconnects, and show that the latter ones scale much better than broadcast networks, which suffer primarily from wire delays resulting from significantly larger area required for wiring. They considered issue widths between 4 and 16, and showed that operand broadcast is not necessary in these architectures. They evaluated the interconnects for conventional VLIWs and Grid Processors [20]. Terechko *et al*. also analyze interconnect models for a clustered VLIW architecture [33], and Peh *et al*. analyze several interconnects for CMP-like architectures such as Raw and TRIPS [36], and propose several power saving techniques. Finally, Taylor *et al*. analyze a static interconnect

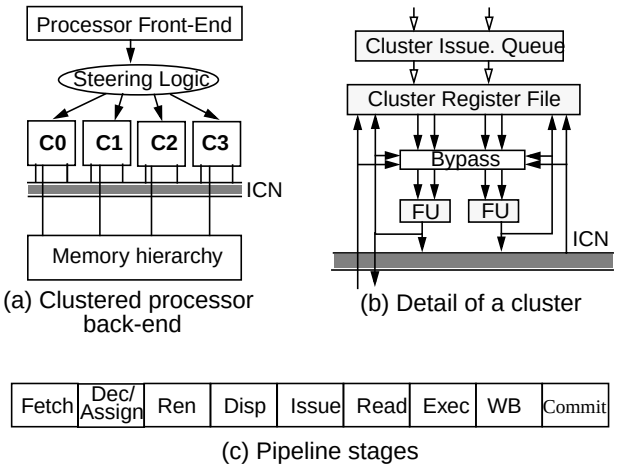


Figure 1. Clustered microarchitecture

model for the MIT's Raw machine [31]. These works mainly differ from ours because they consider architectures with a large number of clusters where cluster assignment and/or scheduling of instructions is performed at compile time, whereas our work focuses on a clustered superscalar microarchitecture with much fewer clusters and a higher emphasis on the network impact on cluster complexity. Finally, our work significantly differs from traditional research on multi-hop networks in that we use synchronous communication among routers, thus drastically reducing router complexity and communication latency with respect to asynchronous routers.

3. Microarchitecture Overview

Several clustered superscalar microarchitectures with different code partitioning strategies have been proposed [28]. They partition the code either at branch boundaries [12, 29, 35], or grouping dependent instructions together [8, 9, 11, 15, 21, 22]. The microarchitecture assumed in this paper is based on a dependence-based paradigm with a distributed register file [9, 24, 38]. Its instruction steering heuristic focuses on minimizing the penalty produced by inter-cluster communications while keeping the cluster workloads reasonably balanced. Both features are detailed below.

We assume a superscalar processor with register renaming based on a set of physical registers, and an instruction issue queue that is separated from the reorder buffer (ROB), as in the MIPS R10000 [37] or the Alpha 21264 [13] processors. The execution core is partitioned into several homogeneous clusters, each one having its own instruction queue, a set of functional units, and a physical register file (see figure 1). The main architectural parameters of a four-cluster architecture are listed in table 1.

Table 1. Default machine parameters for four clusters

Parameter		Configuration
I-cache L1		32KB, 64-byte line, direct mapped, 1cycle hit
Branch Predictor		Hybrid gshare/bimodal: Gshare has 16-bit global history plus 64K 2-bit counters. Bimodal has 2K 2-bit counters, and the choice predictor has 1K 2-bit counters
Num. clusters (C)		4
Per cluster	Phys. regs.	56 int + 56 fp
	IQ size	16
	Issue width	2
	F.U.	2 int ALU,1 int mul/div,1 fp ALU,1 fp mul
Fetch/Decode width		8
ROB size		128
LSQ size		64
Issue		Out-of-order issue. Loads may issue when prior store addresses are known
D-cache L1		64KB, 64 byte line, 2way set-associative, 3 cycle hit time, 3 R/W ports
I/D-cache L2		256KB, 64 byte line, 4 way assoc, 10 cycle hit
DRAM latency		100 cycles

Since our focus is on cluster interconnects, we assumed a simple centralized front-end and data cache, although some strategies are currently being investigated to distribute these components as well [25]. Also, for simplicity, we have not considered the partitioning into heterogeneous clusters, which might be used to avoid replication of rarely used functional units such as multipliers or FP units, or to reduce path length and connectivity to memory ports. Anyway, the techniques proposed in this paper can easily be generalized for heterogeneous clusters.

3.1. The Distributed Register File

The steering logic determines the cluster where each instruction is to be executed, and then the renaming logic allocates a free physical register from that cluster to its destination register. The renaming map table dynamically keeps track of which physical register and cluster each logical register is mapped to, and it has space to store as many mappings per logical register as clusters. Register values are replicated only where they are needed as source operands. When a logical register is redefined with a new mapping, all previous mappings of the same logical register are cleared and saved in the reorder buffer (ROB), to allow freeing the corresponding physical registers at commit time.

Since the physical register file is distributed, source and destination registers are only locally accessed within each cluster. A register value is only replicated in the register file of another cluster when it is required by a subsequent dependent instruction to be executed in that cluster. In that case,

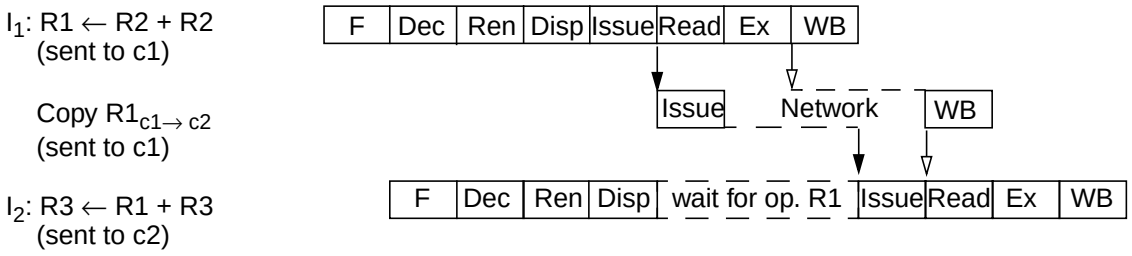


Figure 2. Sample timing of a communication between two dependent instructions I_1 and I_2, steered to clusters c1 and c2 respectively (solid arrows mean wakeup signals, hollow arrows mean data signals, and transmission time is 2 cycles in both cases).

the hardware automatically generates a special *copy instruction* to forward the operand (see figure 2) that will logically precede the dependent instruction in program order. The *copy* is inserted into both the ROB and the instruction queue of the producer's cluster, and it is issued when its source register is ready and it secures a slot in the network. Then, it reads the operand either from the register file or the bypass, sends it through the interconnection network, and delivers it to the consumer's cluster bypass network and register file. The copy also sends through the network, along with the value, the tag of the destination physical register, in order to wake-up the dependent instructions.

Copy instructions are handled just like ordinary instructions, which helps simplifying the scheduling hardware and keeping exceptions precise, although they must follow a slightly different renaming procedure: a free physical register is allocated in the destination cluster, and this mapping is noted in the map table's entry corresponding to the logical register but, unlike ordinary instructions, the old mappings are not cleared.

4. Improved Steering Schemes

Our baseline cluster assignment algorithm is a variation of the data-dependence scheme proposed by Parcerisa and González [24], who showed that it is the one achieving the best performance for this microarchitecture [23]. This scheme (summarized in figure 3) follows primary and secondary criteria, and works in the following way. In normal operation, i.e. when there is no workload imbalance, the primary criterion (labelled as rule 1) first selects the clusters that minimize communication penalties, then if more than one cluster is selected, the secondary criterion (labelled as rule 2) chooses the least loaded one. However, in situations when the workload imbalance exceeds a

```
                if (workload_imbalance < threshold)
                {
rule 1.             if (any source register is not available)
1.1.                    select the producer cluster
                    else
1.2.                    select the clusters with the highest no. of source registers mapped
                }
rule 2.         choose the least loaded of the above selected clusters
```

Figure 3. Rules of the Baseline Steering Algorithm

given threshold, the primary criterion is ignored. To satisfy the primary criterion (i.e., to minimize communication penalties), the heuristic distinguishes two cases: first (rule 1.2), if any of the source registers is not available, it chooses the cluster where it is going to be produced; and second (rule 1.2), if all the source registers are available, it chooses the clusters with the highest number of source registers mapped.

Regarding the workload imbalance estimation, we assumed the DCOUNT metric which may be defined, from a conceptual standpoint, as the maximum of the absolute deviations of the accumulated number of dispatched instructions per cluster. This metric may be easily implemented with one signed counter per cluster which computes the difference between the number of instructions dispatched to that cluster and the average for all clusters (refer to [23, 24] for further details). These counters are updated once per cycle, and they are cleared on a branch misprediction recovery. The mentioned threshold was empirically set to 32 and 64, for 4 and 8 clusters, respectively.

4.1. Reducing Communications with Accurate-Rebalancing (AR) Steering

One major drawback of the above baseline cluster assignment algorithm is that it generates too many communications during the periods when the workload imbalance exceeds the threshold, because in such cases, it totally ignores dependences. Moreover, the probability that the steering algorithm generates a communication in such situations grows with the number of clusters. Since we are going to analyze configurations with four and eight clusters, it makes sense to find a more accurate method for rebalancing the workload without generating as many communications.

We observed that most often, a strong imbalance situation is caused by a single overloaded cluster. Of course, rebalancing the workload does require to steer instructions to the less loaded clusters, but choosing strictly the least loaded one is probably not the best solution. For a two-clustered organization there is no other alternative, but for more clusters the steering scheme could recover the

(a)

```
if (workload_imbalance >=threshold)
    exclude clusters with workload > 0

rule 1.  if (any source register is not available)
1.1.        select the producer cluster
         else
1.2.        select the clusters with the highest no. of source registers mapped

rule 2.  choose the least loaded of the above selected clusters
```

(b)

```
if (workload_imbalance >=threshold)
    exclude clusters with workload > 0

rule 1.  if (any source register is not available)
1.1.        select the producer cluster
         else
1.2.        select the cluster that minimizes the longest communication distance

rule 2.  choose the least loaded of the above selected clusters
```

Figure 4. Rules of the Accurate Rebalancing Steering (a), and the Topology-Aware Steering (b)

strongly imbalanced situation with more accurate rebalancing actions that do not ignore the dependences completely, and thus generate less communications.

Therefore, we propose to improve the baseline scheme with the Accurate Rebalancing (AR) technique. This scheme (summarized in figure 4a) works in the following way. In case of a strong imbalance situation, instead of directly choosing the least loaded cluster, the algorithm follows the two baseline criteria except that the most loaded clusters are previously excluded from the choice of clusters. In doing so, there is a chance that among the non excluded clusters there is one where the source registers are mapped and thus inter-cluster communications are not required. We experimented with different exclusion criteria based on the existing signed workload counters and we found the most simple and effective one is to exclude clusters that have a positive workload counter.

4.2. Reducing Communication Latency with Topology-Aware (TA) Steering

For many of the interconnect topologies we study in this paper, the latency of the communications depends on the distance between source and destination clusters. A topology-aware (TA) steering heuristic can take advantage of this knowledge to minimize the distance -and thus the latency- of the communications. Therefore, we have refined the primary criterion to take the distance into account. This algorithm (see figure 4b) in the case that all source operands are available (rule 1.2),

chooses the clusters that minimize the longest communication distance (the one that is in the critical path). To illustrate this feature, let us suppose that an instruction has two source operands, which are both available, and the left one is mapped to cluster 1, while the right one is mapped to clusters 2 and 3. In this case, the original primary criterion would select clusters 1, 2, and 3, since all of them have one operand mapped. Whatever is chosen, one copy would be needed, either between clusters 1 and 2 or between clusters 1 and 3. If we assume that cluster 1 is closer to cluster 2 than to cluster 3, then the topology-aware heuristic will consider only clusters 1 and 2.

5. The Interconnection Network

In this section we discuss several design trade-offs and constraints regarding the interconnection network, prior to describing in detail, in the following two sections, the models that have been experimentally analyzed for architectures with four and eight clusters.

5.1. Synchronous versus Asynchronous Communication

Interconnection networks have been widely studied in the literature for different computer areas such as multicomputers and networks of workstations (NOWs) [10]. In these contexts, communication latencies may be thousands of processor cycles long. Moreover, it is unfeasible to distribute a single clock signal among all the processors. Thus, communication between processing nodes is asynchronous, which requires a large buffering area at each router input link and a relatively complex router design [27]. In contrast, for clustered microarchitectures performance is highly sensitive to the communication latency and just one cycle is a precious time, as shown by the results in section 8, and also by other previous works [5, 8]. Thus, in this context, router design should be kept as simple and fast as possible. In particular, taking into account that it is feasible to distribute a single clock signal among all the routers, we propose using synchronous communication among them. This will drastically simplify router design and eliminate the need for large buffers. Also, networks must use simple routing schemes that carefully minimize communication latency (instead of maximizing throughput, like in other contexts). We assume that all routing decisions are locally made at issue time (source routing), by choosing the shortest path to the destination cluster. If there is more than one minimal route, the issue logic chooses the first one that it finds available.

5.2. Register File Write Ports

Each cluster can inject copies into the network, which connects the cluster register files through a number of dedicated write ports where copies are delivered. From the point of view of the network design, including as many ports as required by its peak delivery bandwidth is the most straightforward alternative, but the number of write ports has a high impact on cluster complexity. First, each additional write port requires an additional result tag to be broadcast to the instruction issue queue, and the wakeup delay increases by a quadratic factor with respect to the number of broadcast tags [21]. Second, the register file access time increases linearly with the number of ports. Third, the register file area grows quadratically with the number of ports, which in turn makes the length and delay of the bypass wires to increase.

Moreover, previous studies showed that, with adequate steering heuristics, the required average communication bandwidth is quite low (around 0.22 communications per instruction, for 4 clusters [24]), and thus it is unlikely that having more than one write port per cluster connected to the network can significantly improve performance. Therefore, for all the analyzed networks we assume that they are connected to a single write port per cluster, except for the idealized models.

5.3. Communication Timing

In our distributed register file architecture, the access to remote operands is done exclusively through copy instructions, which are inserted into the instruction queues as normal instructions. A copy is issued when its source register is ready and it secures a slot in the network. Then, it reads the operand either from the register file or from the bypass, sends the value through the interconnection network, and delivers it to the consumer's cluster bypass network and register file.

As mentioned above the copy sends the tag of the destination physical register one cycle ahead of the value. We assumed for simplicity that the tag forwarding delay is the same as the data forwarding delay. Consequently, the tag forwarding stays in the critical path of execution of the dependent instruction, which also includes issuing the copy instruction (see figure 2). Therefore, the total minimum issue distance between the producer and the consumer instructions equals the communication latency plus one cycle. However, a particular VLSI implementation could attempt to reduce this issue distance by optimizing the tag forwarding paths, which would leave it equal to the data communication latency.

5.4. Transmission Time

The total latency of a communication has two main components: the contention delays caused by a limited bandwidth, and the transmission time caused by wire delays. For a given network design, the first component varies subject to unpredictable hazards, and we evaluate it through simulation. On the other hand, the second component is a fixed parameter that depends on the propagation speed and length of the interconnection wires, which are low-level circuit design parameters bound to each specific circuit technology and design.

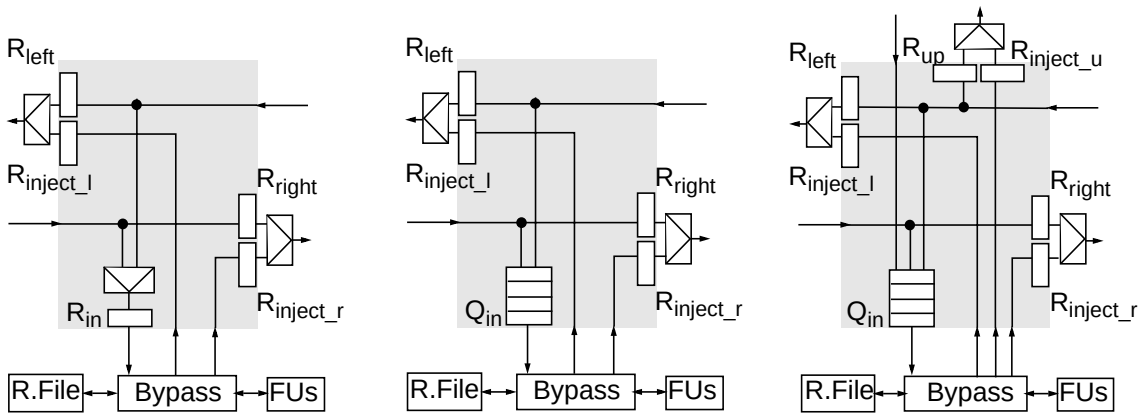
To help narrowing this complex design space, we have taken two reasonable assumptions for point-to-point networks. First, the minimum inter-cluster communication latency is one cycle. This clock cycle includes wire delay and switch logic delay. Note that, with current technology, most of the communication latency is wire delay. Second, only neighbor clusters (those at a one-cycle distance) are directly connected with a pair of links, one in each direction. As a consequence, the communication between two non-neighbor clusters takes as many cycles as the number of links it crosses.

With these two assumptions, the space defined by different propagation speeds and wire lengths is discretized and reduces to the one defined by a single variable: the number of clusters that are at one-cycle distance from a given cluster (which is an upper bound of the connectivity degree of the network). Our analysis covers a small range of this design space by considering the connectivity degrees of several typical regular topologies.

Consistent with these long wire delays, the centralized L1 data cache is assumed to have a 3 cycle pipelined hit latency (address to cache, cache access and data back).

5.5. Router Structures

We assume a very simple router attached to each cluster for point-to-point interconnects. The router enables communication pipelining by implementing stage registers (buffers) in each output link (R_{right}, R_{left}, and R_{up}, in figure 5). To reduce the complexity, the router does not include any other buffering storage for in-transit messages, but it rather guarantees that after receiving an in-transit message, it will be forwarded in the next cycle. This requirement is fulfilled by giving priority to in-transit messages over newly injected ones, and by structurally preventing that two in-transit mes-



(a) Connected to 2 nodes (sync. ring).

(b) Connected to 2 nodes (async. ring).

(c) Connected to 3 nodes (async. mesh and torus).

Figure 5. Router schemes for synchronous and asynchronous point-to-point interconnects

sages compete for the same output link. Such a competence between in-transit messages never occurs on nodes with two neighbors, like those in a ring (figures 5a and 5b) but may happen in other network topologies like a mesh or a torus, where each node may have up to 4 neighbors (note, however, that since we have considered only small meshes with 4 and 8 clusters, each node has never more than 3 neighbors). For nodes with three neighbors, the router constrains the connectivity of in-transit messages (figure 5c) by connecting every input link to a single stage register (output link), in the three ways: in-transit messages can traverse the router from the left to the right link, from the right to the left link or from the right to the upper link. Note that messages arriving from the upper link have no other connection than the input queue, thus this link is only available for messages doing their last hop.

A *copy* instruction is kept in the issue queue until both its source operand is available and it secures the required injection register (R_{inject}, in figure 5), so no other buffering storage is required. That is, the scheduler handles the router injection registers as any other resource. While access requests for a bus-based network are sent to a distant centralized arbiter, the arbitration of each link in a point-to-point network is done locally at the source cluster, by simply choosing between one injection register and one stage register (priority is given to the latter one, as mentioned above). Eventually, the *copy* is issued and the outgoing message stays in one of the R_{inject} output registers while being transmitted through the first hop.

The router also interfaces with the cluster datapath. For partially asynchronous networks, the router includes an input FIFO buffer (Q_{in}, in figures 5b and 5c) where all incoming messages are queued. Each cycle, only the message at the queue head is delivered to the cluster datapath, the others stay in the queue. Writing to this buffer may require control flow to prevent overflows. Solutions to this problem are discussed in section 8.4. For synchronous networks, the router is even less complex. By appropriately scheduling the injection of messages at the source cluster (more details are given later), the proposed scheme guarantees that a given router does not receive more than one input message per cycle. Therefore, the router requires just a single register (R_{in}, in figure 5a), instead of the FIFO buffer.

5.6. Bus versus Point-to-Point Interconnects

Although our analysis mainly focuses on point-to-point networks, we also study a bus interconnect, for comparison purposes. It is made up of as many buses as clusters, each bus being connected to a write port in one cluster, and each cluster being able to send data to any bus (figure 6a). Although this is a conceptually simple model, it has several drawbacks that make it little scalable. First, since buses are shared among all clusters, their access must be arbitrated, which makes the communication latency longer, although bandwidth is not affected as long as arbitration and transmission use different physical wires. Second, a large portion of the total available bandwidth, which is proportional to the number of clusters, is wasted due to the low bandwidth requirements of the system. However, if the number of buses was reduced, then the number of conflicts would increase, and hence the communication latency. Third, each bus must reach all clusters, which implies long wires and long transmission times, which can drastically reduce the bandwidth if the bus transmission time is not pipelined[1].

Compared to the above bus interconnect, a point-to-point interconnect (a ring, a mesh, a torus, etc.) has the following advantages. First, the access to a link can be arbitrated locally at each cluster. Second, communications can be more easily and effectively pipelined. Third, delays are shorter, due to shorter wires and smaller parasitic capacitance (there are less devices attached to a point-to-point link than to a bus). Fourth, network cost is lower than a configuration with as many buses as

1 Note that it is difficult to pipeline bus communications, but it is easy to pipeline communication through point-to-point links (although the latter case is not needed with current VLSI technology, so we assume a transmission time of 1 cycle per hop), which clearly indicates that point-to-point links are much more scalable than buses.

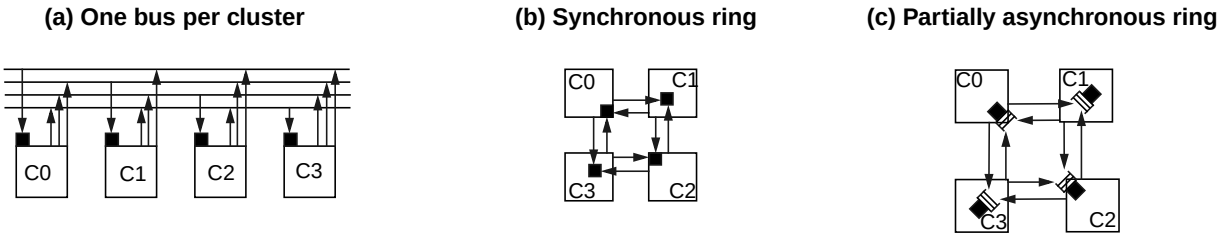


Figure 6. Four-cluster topologies

clusters. Finally, it is more scalable: when the number of clusters increases, its cost, bandwidth and ease of routing scales better than for the bus-based configuration.

6. Four-Cluster Network Topologies

For four clusters, we propose two alternative point-to-point networks based on a ring topology, and compare them to a realistic bus-based network (see figure 6). We also compare their performance to that of an idealized ring, which represents an upper bound for ring networks. Below we describe these topologies.

6.1. Bus2

This is a realistic bus interconnect with a 2-cycle transmission time (hence its name). It has as many buses as clusters, each one connected to a single write port (see figure 6a), and a very simple centralized bus arbiter (one per bus). The total communication latency is 4 cycles because bus arbitration, including the propagation of the request and grant signals, takes 2 additional cycles. We assume that the arbitration time may overlap with the transmission time of a previously arbitrated communication, so each single bus bandwidth is 0.5 communications per cycle.

6.2. Synchronous Ring

This interconnect is one of the contributions of this work, since previously proposed rings work in asynchronous mode. This topology assumes no queues in the routers, neither to store in-transit messages nor to store messages arriving at their destination clusters.

Since no queues are included at the destination clusters, when a message arrives it must be immediately written into the register file. The router arbitration logic injects copy instructions with an algorithm (summarized in table 2) that ensures that no more than one message arrives at a time at a given node, since it forces messages in clockwise direction to arrive at destination during an odd

cycle, while those in counter-clockwise direction arrive during an even cycle. Other conflicts are avoided by just giving priority to in-transit messages over newly injected ones, so they are never stalled at the intermediate nodes. During odd cycles, a source cluster *src* is allowed to send a one-hop message (D=1) to its adjacent cluster in the clockwise direction (*(src +1) mod 4*), and a two-hops message (D=2) in the counter-clockwise direction (*(src + 2) mod 4*). During an even cycle, the allowed directions are reversed. Since in-transit messages are given priority over newly injected ones (see section 5.5), an issued copy instruction may have to wait in the injection register until the cycle parity is appropriate.

Table 2. Rules to inject a message at the source cluster *src* (D refers to distance in cycles)

Direction	Odd Cycle		Even Cycle	
	D	Destination Cluster	D	Destination Cluster
Clockwise	1	*src → (src+1) mod 4*	2	*src → (src+2) mod 4*
Counter-clockwise	2	*src → (src+2) mod 4*	1	*src → (src+3) mod 4*

Despite the fact that there are cyclic dependencies between links [10], deadlocks are avoided by synchronously transmitting messages through all the links in the ring, even if the stage buffer at the next router is busy (it will be free when the message arrives). This is possible thanks to using the same clock signal for all the routers and giving a higher priority to in-transit messages.

6.3. Partially Asynchronous Ring

Typical asynchronous networks include buffers both in the intermediate routers, to store in-transit messages, and in the destination routers, to store messages that are waiting for a write port (in our case to the register file) [10]. The former are removed in our design, like in the synchronous ring. However, we still need the latter, since two messages may arrive at the same time to the same destination cluster and there is only one write port in each cluster. In this case, the message whose data cannot be written is delayed until it has a port available. Note that the system must implement an end-to-end flow control mechanism in order not to lose messages when a queue is full. This is an additional cost of the asynchronous schemes, which is discussed in more detail in section 8.4. In this network, routers use the same clock signal. Therefore, it is only partially asynchronous. A fully asynchronous network has not been considered because its cost would be much higher (larger buffers, link-level flow control, extra buffers to avoid deadlocks, etc.). Deadlocks are avoided as in the synchronous ring.

6.4. Idealized Ring and Crossbar Topologies

For comparison purposes, we select two idealized topologies: a ring and a crossbar. In the idealized ring, inter-cluster distances are the same as those of the realistic ring (discussed above), but an unlimited bandwidth is assumed, which makes it contention-free (i.e., it has an unlimited number of links between each pair of nodes and an unbounded number of register file write ports for incoming messages in each cluster). Therefore, this idealized topology lets us estimate how much performance is lost due to interconnect bandwidth constraints.

On the other hand, the idealized crossbar assumes that every cluster is at 1-cycle distance of each other, in addition to the unlimited bandwidth assumed by the ideal ring. Therefore, its performance is an upper bound for all other models, and it allows us to estimate how much performance is lost due to constraining the connectivity degree -and hence the complexity- to 2 adjacent nodes per cluster.

7. Eight-Cluster Network Topologies

For eight-cluster architectures, we first consider two ring-based interconnects, synchronous and partially asynchronous, similar to those proposed for 4-cluster architectures, and also two versions of a realistic bus-based network, having transmission times of 2 and 4 cycles, respectively.

In addition to the ring, we also analyze mesh and torus topologies, both of them partially asynchronous, since they feature lower average communication distances. Figure 7 shows these three schemes. Below, we describe each scheme in detail.

7.1. Bus2 and Bus4

The bus required to connect 8 clusters is likely to be slower than that required by the 4-cluster configuration due to longer wires and higher capacitance. To account for this, we consider two bus-based configurations: the Bus2, which optimistically assumes the same latencies as those of the 4-cluster configuration (i.e., a transmission time of 2 cycles), and the Bus4, which more realistically assumes twice this latency (i.e., a transmission time of 4 cycles).

7.2. Synchronous and Partially Asynchronous Rings

The partially asynchronous ring model is analogous to that described for 4 clusters (section 6.3).

For the synchronous ring (see figure 7a), the router arbitration logic injects copy instructions with an algorithm (summarized in table 3) analogous to the one discussed in section 6.2 for 4 clusters. Likewise, this algorithm prevents two messages from arriving at once to the same cluster because it forces messages in clockwise direction to arrive at destination during an odd cycle, while those in counter-clockwise direction arrive during an even cycle.

Table 3. Rules to inject a message at the source cluster *src* (D refers to distance in cycles)

Direction	Odd Cycle		Even Cycle	
	D	Destination Cluster	D	Destination Cluster
Clockwise	1	*src → (src+1) mod 8*	2	*src → (src+2) mod 8*
	3	*src → (src+3) mod 8*	4	*src → (src+4) mod 8*
Counter-clockwise	4	*src → (src+4) mod 8*	3	*src → (src+5) mod 8*
	2	*src → (src+6) mod 8*	1	*src → (src+7) mod 8*

7.3. Mesh

A mesh topology (see figure 7b) reduces some distances with respect to a ring. The average distance in a ring is 2.29 hops, while in a mesh it is 2 hops; however, the maximum distance is still 4 hops. The dashed lines in the figure show the links added to the ring topology to convert it into a mesh.

Due to the increased connectivity, this topology introduces a new problem to the design of the routers with respect to a ring, because at central nodes (labelled C2, C3, C6, and C7) more than one in-transit message at the router input links could compete to access the same output link. As discussed in section 5.5, our approach is to constrain the connectivity of in-transit messages within the router. On the one hand, an *upper* router input (refer to figure 5c) is only connected to the input queue, so the routing algorithm must ensure that this link is used only for the last hop of a transmission. The four links between C2-C3 and C6-C7 in our mesh (shown with dashed arrows in figure 7b) are connected to *upper* router inputs and outputs. Thus, if one message is sent, for instance, from cluster C2 to C7, it must be routed through C6 because the link C2-C3 is not connected to the link C3-C7. On the other hand, in-transit messages arriving at a *right* router input have no constraints, so they may be routed either to the *left* or to the *upper* output. The four links between C3-C7 and C2-C6 in our mesh (figure 7b) are connected to *right* router inputs and outputs. Finally, the rest of links connected to the four central nodes of the mesh are connected to *left* router inputs. In-transit messages arriving to a *left* router input can only be routed to the *right* output. Thus, for instance, a message from C0 to C3 must be routed through C1 because the link C0-C2 is not connected to the link C2-C3.

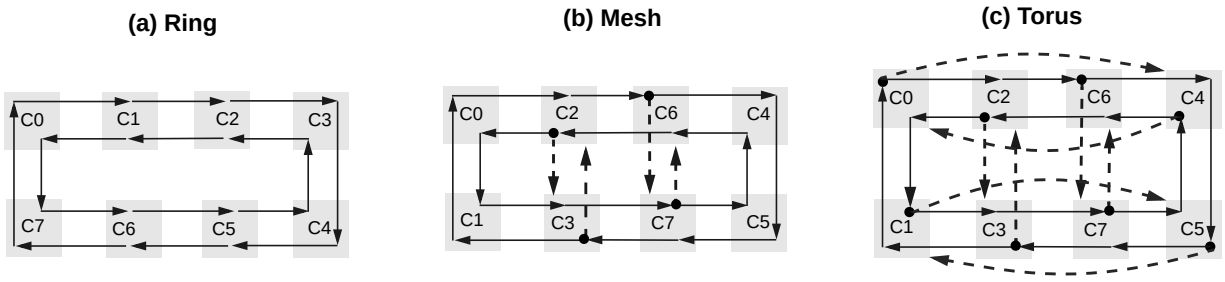


Figure 7. Topologies for 8 clusters. Notice the different node numbering for a ring. Dashed links are constrained to be used only for the last hop of a communication, because of router connectivity constraints.

Deadlocks within the rings (solid links in figure 7.b) are avoided by using the same clock signal for all the routers and transmitting messages synchronously. The remaining links (dashed links in figure 7.b) cannot contribute to any deadlock because they are used only for the last hop.

7.4. Torus

A torus has smaller average distance than a mesh (see figure 7c). At each node, more than one in-transit message at the router input links could compete to access the same output link. Like for the mesh, this problem is solved without including intermediate buffers, by constraining the connectivity of several links. The solution is outlined in the figure, where dashed arcs indicate links with a limited connectivity (see also router details in figure 5c).

Note that this constraint does not change the minimal distance between every pair of nodes, but for some pairs it does reduce the number of alternative routes. For example, there is only one two-hops route from C4 to C1. However, due to the poor utilization of the network (as results will show later) this is a minor drawback.

Again, deadlocks are avoided as indicated above for the mesh. On another issue, when mapping torus links on silicon, some links may be longer than the rest (e.g., links between C0 and C4). This may introduce delays in those particular links. For the sake of simplicity, we did not consider that additional delay, which can be avoided by interleaving the routers in each dimension [10].

7.5. Idealized Torus and Crossbar Topologies

Similarly to four-cluster topologies, we select two idealized topologies for comparison: a torus and a crossbar. Distances in the idealized torus are identical to those of the realistic torus but with unlimited bandwidth, which makes the network contention-free. In other words, it is assumed that the

network has an unlimited number of links between any pair of adjacent nodes and an unbounded number of register file write ports connected to the network in each cluster. Therefore, its performance is an upper bound on the performance of the realistic torus.

In the idealized crossbar, in addition to the unlimited bandwidth, all clusters are at 1-cycle distance of each other. In this case, its performance is an upper bound for all other models, and it allows us to gauge how much performance is lost due to constraining the connectivity degree.

8. Experimental Results

In this section, the simulation environment is first described and then the different network architectures proposed above are evaluated.

8.1. Experimental Framework

To perform our microarchitectural timing simulations, we have extended the sim-outorder simulator of the SimpleScalar v3.0 tool set [7] with all the architectural features described above, including the different interconnection network topologies.

For these experiments, we have used the Mediabench benchmark suite [18]. This benchmark suite captures the main features of commercial multimedia applications, which are a growing segment of commercial workloads. All the benchmarks were compiled for the Alpha AXP using Compaq's C compiler with the -O4 optimization level, and they were run till completion. For the sake of completeness, we have also run the same experiments with the SpecInt95 benchmark suite (results are reported in section 8.7).

All topologies maintain the same processor model. Table 1 summarizes the machine parameters of a four-cluster architecture used through the simulations. The eight-clusters architecture assumed an identical cluster model as those of the four-clusters architecture, which means doubling the total effective issue width of the processor. Accordingly, the eight-cluster architecture also assumes twice the fetch/decode bandwidth, number of data cache ports and number of entries in the reorder buffer and in the load/store queue.

8.2. Network Latency Analysis

To gain some insight on the different behavior of synchronous and partially asynchronous rings for a 4-cluster architecture, we analyze their average communication latency. In particular, since the

transmission time component of the latency is the same for both interconnects, we only analyze the contention delay component.

Figure 8 compares the communications contention delay for each of the two ring interconnects, and it also includes an ideal ring for comparison. For each of the two former interconnects, the contention delay has two components: it may be caused by an insufficient issue width or an insufficient interconnect bandwidth. In contrast, the contention delay of the ideal ring is exclusively due to the limited issue width, and it is on average 0.8 cycles. Therefore, comparing the delays of the first two interconnects to that of the ideal ring, the difference gives an estimation of the contention caused by the insufficient interconnect bandwidth.

As shown in figure 8, one-hop messages (graph a) wait for longer than two-hops ones (graph b). The main reason is the available bandwidth for each type of message: the latter have two alternative minimal-distance routes, while the former have only one. However, since the routing algorithm is the same for both ring interconnects, it does not explain the differences observed between the two rings.

Figure 8a shows that the contention delay of one-hop messages for a synchronous ring (2.19 cycles) is two times longer than for a partially asynchronous one (1.06 cycles). In contrast, the contention caused by two-hops messages for a synchronous ring (0.72 cycles) is just slightly lower than for a partially asynchronous one (0.87 cycles). These differences are due to the different ways each interconnect avoids conflicts between messages that require access to the same register file write port: in a synchronous ring, these conflicts are prevented by ensuring that a one-hop message is not issued if the parity of the cycle is not the appropriate one (see table 2), regardless of whether the link is busy or not. For example, a two-hops message from C1 to C3 (see figure 6b) will be injected in an even cycle, thus reaching the router at C2 and requesting the C2-C3 link during the next odd cycle. As messages from C2 to C3 must be injected during odd cycles, these one-hop messages will be delayed if there are in-transit two-hops messages. In a partially asynchronous ring, a message of any kind can be issued as soon as the required output link is available, although it may have to wait at the destination cluster router until it gains access to the register file write port. For an asynchronous ring, the average contention delays caused by the network to one-hop and two-hops messages are 0.14 and 0.22 cycles respectively, whereas the rest of the contention delay of the communications is due to the issue width.

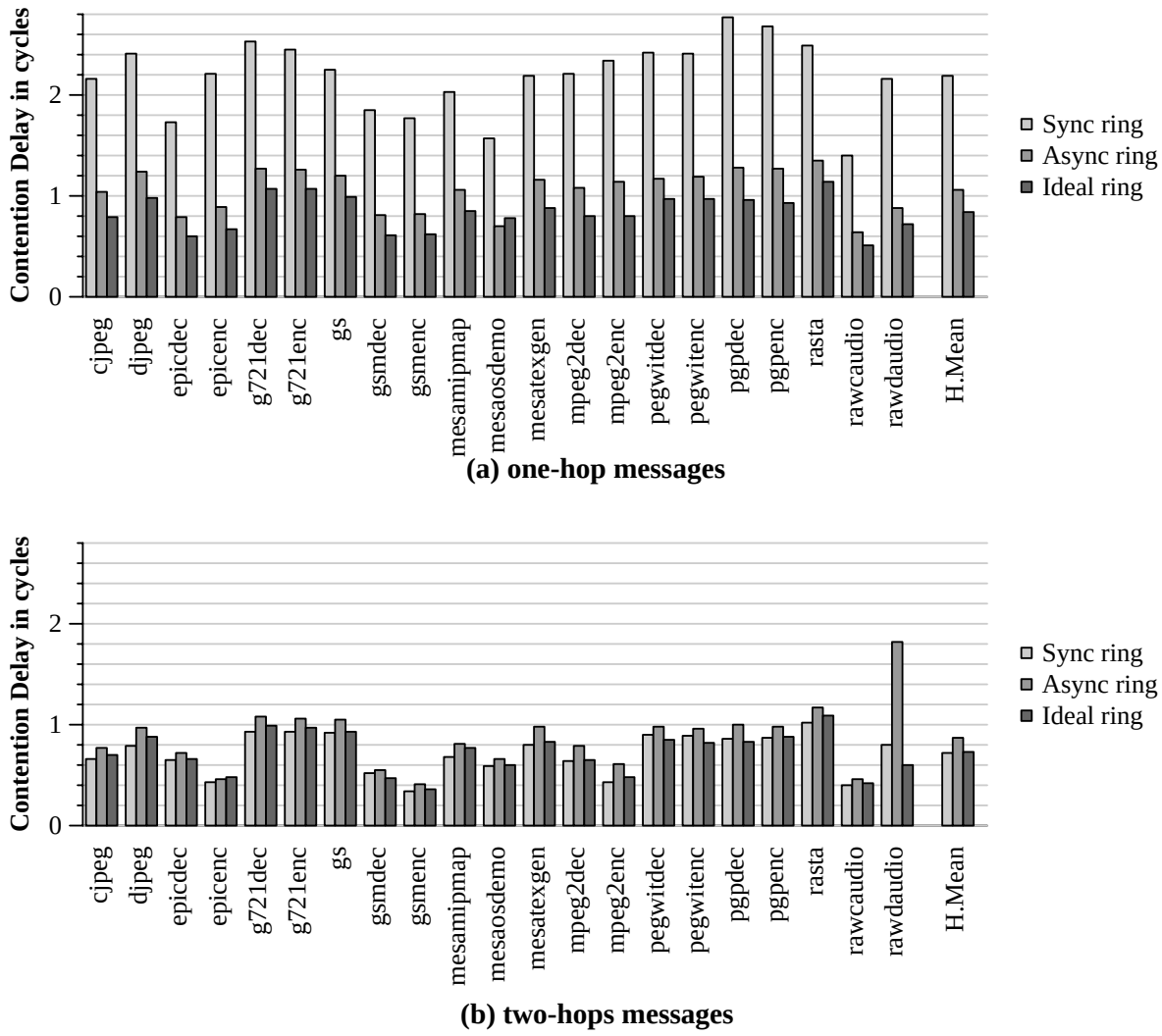


Figure 8. Average contention delays of one-hop and two-hops messages, with synchronous, partially asynchronous and ideal ring interconnects

To summarize, the long delays caused to one-hop messages by the scheduling constraints of the synchronous ring make its overall contention delay be higher than for a partially asynchronous one. In addition, since there are about twice as many one-hop messages as two-hops ones, they have a high impact on the overall contention delay. As a consequence, the partially asynchronous ring performs better than the synchronous one, as it is shown below.

8.3. Performance of Four-Cluster Interconnects

Figure 9 compares the performance, reported as number of committed instructions per cycle (IPC), of a four-cluster architecture for all the proposed and referenced interconnects.

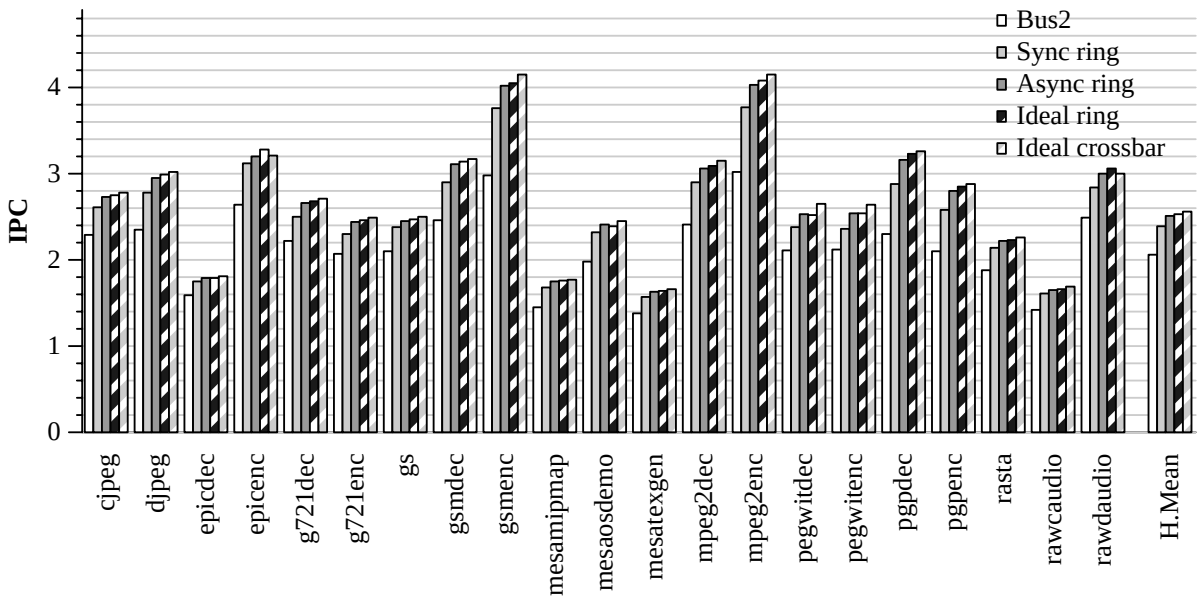


Figure 9. Comparing the IPC of 4-cluster interconnects

The two ring interconnects consistently achieve better performance than the bus topology, for all benchmarks. This is mainly because one-hop messages have a shorter transmission time in point-to-point interconnects, and because the steering heuristic exploits it to keep close instructions that have to communicate. Besides, the ring topology offers a higher bandwidth, although in this scenario bandwidth is not critical for performance due to the low traffic generated by the steering scheme [24] (e.g., it is on average 0.20 communications per instruction, for a partially asynchronous ring).

The IPC achieved by the synchronous and partially asynchronous rings is, on average, 15.8% and 21.7% higher, respectively, than that achieved by the bus. This is because the contention delays of one-hop messages are lower for the asynchronous ring, as discussed in section 8.2.

The performance of the partially asynchronous ring is very close to that of the ideal ring (less than 1% difference), which shows that increasing the number of links or the number of register file write ports would hardly improve performance. In other words, due to the effectiveness of the steering logic to keep the traffic low, a simple configuration with two links between adjacent clusters (one in each direction) and a single register file write port for incoming messages is clearly the most cost-effective design.

Finally, we found that a ring performs very close to a complex fully connected crossbar. The performance lost by reducing the connectivity degree from 3 to 2 adjacent nodes per cluster (cor-

responding to the ideal crossbar and the ideal ring, respectively) is on average just 1.4%. Therefore, we can conclude that a ring offers a good cost-performance ratio for a four cluster interconnect.

8.4. Queue Length

Asynchronous rings need specific mechanisms to prevent (or to recover from) potential overflows of the network buffers. In our partially asynchronous interconnect, this problem occurs only in the queues for incoming messages at each cluster.

In order to adequately dimension these queues, we first assumed unbounded size queues and measured the number of occupied entries each time a new message arrives at its destination cluster. Note that with FIFO queues and a single write port, this number is equal to the number of cycles a message stays in the queue. We found that for any benchmark, more than 85% of the messages do not have to wait because they find the queue empty (92.1%, on average), and the maximum observed number of occupied entries was 11. For instance, table 4 shows a typical queue length distribution (for benchmark *djpeg*).

Table 4. Queue length distribution (for *djpeg*)

# occupied entries	# messages	Distribution (% times)	Cumulative Distribution (%)
0	1263097	90.29	90.29
1	122410	8.75	99.04
2	11973	0.86	99.89
3	1376	0.10	99.99
4	108	0.01	100.00
5	14	0.00	100.00
>=5	4	0.00	100.00

Although 11-entry queues are long enough in our experiments, the model should ensure that data is never lost, in order to guarantee execution correctness. Two approaches are possible: first, to implement a flow control protocol that prevents FIFO queue overflows; and second, to implement a recovery mechanism for these events. Flow control can be based on credits. In this case, each cluster would contain a credit counter for each destination cluster. Every time a message is transmitted to a cluster, the corresponding credit counter would be decreased. If the counter is equal to zero, the message would not be transmitted because the FIFO queue may be full. When a message is removed from the queue, a credit is returned to the sender of that message, thus, consuming link bandwidth. Upon reception of the credit, the corresponding credit counter is increased. However, since overflows are so infrequent, the most cost-effective solution in case of an overflow is to

squash the instruction that generated the message that caused the overflow, as well as all other younger instructions, very much like in the case of exceptions or branch mispredictions, and to re-start again execution at this instruction. This approach requires minimal additional hardware and it produces negligible performance penalties (for 11-entry queues there is no penalty at all for our benchmarks).

8.5. Performance of Eight-Cluster Interconnects

In this section, we evaluate the eight-cluster network interconnects described in section 7, for a 16-way issue architecture (as described in section 8.1). Figure 10 shows the IPC for the different schemes. The point-to-point ring achieves a significant speed-up even over the optimistic bus architecture denoted as bus2. The average speed-up of the synchronous ring over bus2 is 12.1% whereas the partially asynchronous ring outperforms bus2 by 19.3%.

Comparing the partially asynchronous topologies, the mesh achieves an IPC 2.1% higher than that of the ring, while the IPC of the torus is 4.9% higher than that of the ring. On the other hand, the partially asynchronous torus performance is very close to that of the ideal torus configuration with unlimited bandwidth (less than 1% difference). The performance of the ideal torus is just 3.4% below that of the ideal crossbar that has unlimited bandwidth and all nodes at a 1-cycle distance.

8.6. Effectiveness of the Accurate-Rebalancing and Topology-Aware Steering

In all the previous experiments it was assumed that both the Accurate-Rebalancing (AR) and the Topology-Aware (TA) improvements described in section 4 were active. In this section, the effectiveness of these two techniques are analyzed. They are evaluated for a four-cluster architecture with an asynchronous ring and for an eight-cluster architecture with a torus interconnect. Figure 11 compares the baseline steering with and without these two techniques. The figure shows the average IPC (graph a), and the average communications rate (graph b) and distance (graph c).

The AR technique is aimed at reducing the communications generated during strong imbalance situations, when the steering is mainly concerned on rebalancing the workload. Instead of totally ignoring dependences, it just excludes the overloaded clusters. As shown in figure 11a, AR improves the performance over the baseline steering by 2.3% and 5.8% for four and eight clusters, respectively, because it significantly reduces the amount of communications. As shown in figure

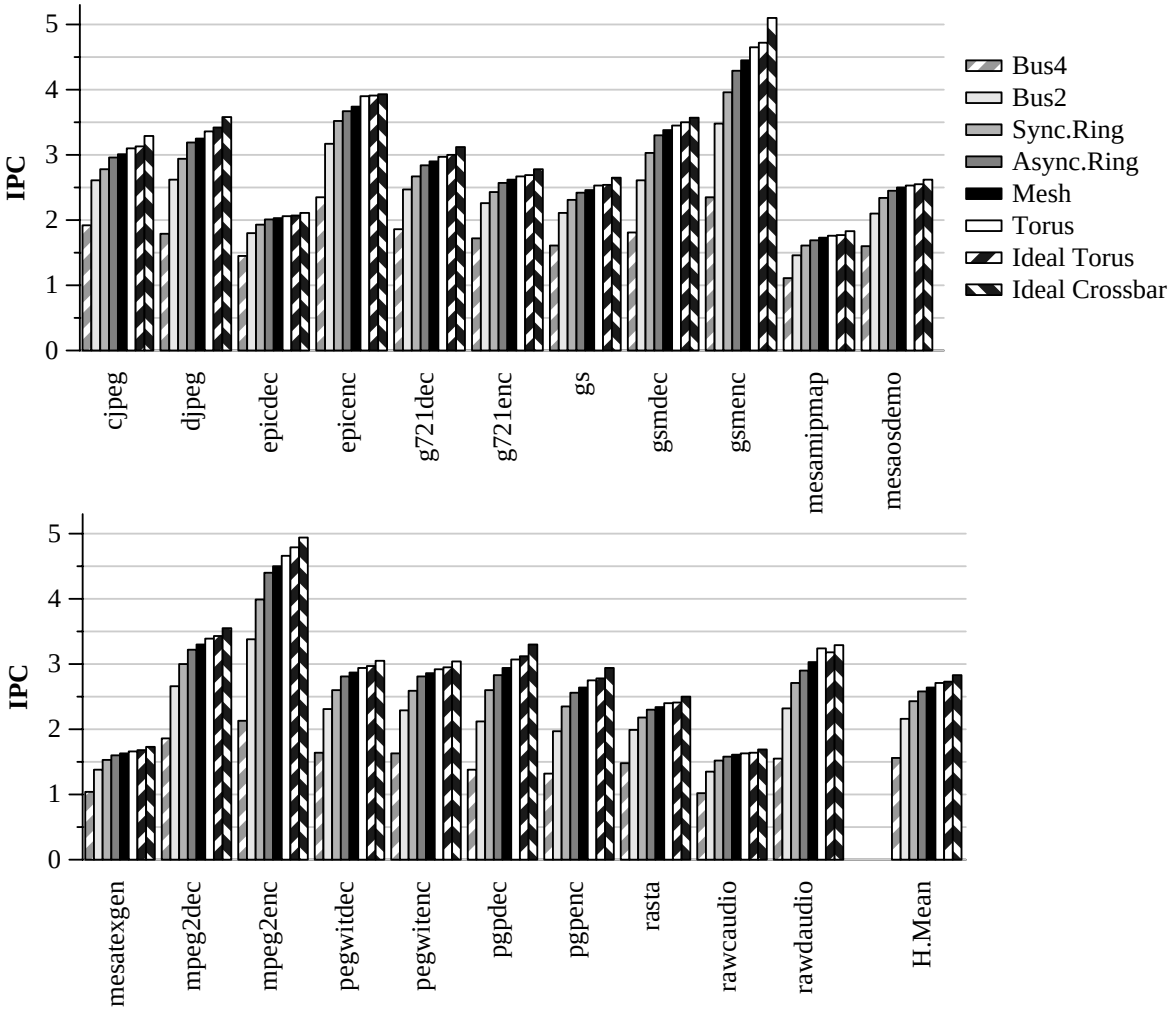


Figure 10. Comparing the IPC of 8-cluster interconnects

11b, AR reduces the communication rate by 13% and 27% for four and eight clusters, respectively. Not surprisingly, the effectiveness of the AR technique grows with the number of clusters, because the likelihood of causing a communication when operand locality is ignored increases with the number of clusters.

The TA technique is aimed at minimizing communication distances -hence latencies- for point-to-point interconnects. As shown in figure 11a, it produces a small 1% IPC improvement over the AR scheme for eight clusters and almost no effect for four clusters (the total improvement using both AR and TA techniques is 7.3% and 2.4%, respectively). TA produces a small impact on performance because there are actually few instructions that offer the chance to reduce the communication distance, and because in some of these cases the distance is reduced at the expense of generating one extra communication.

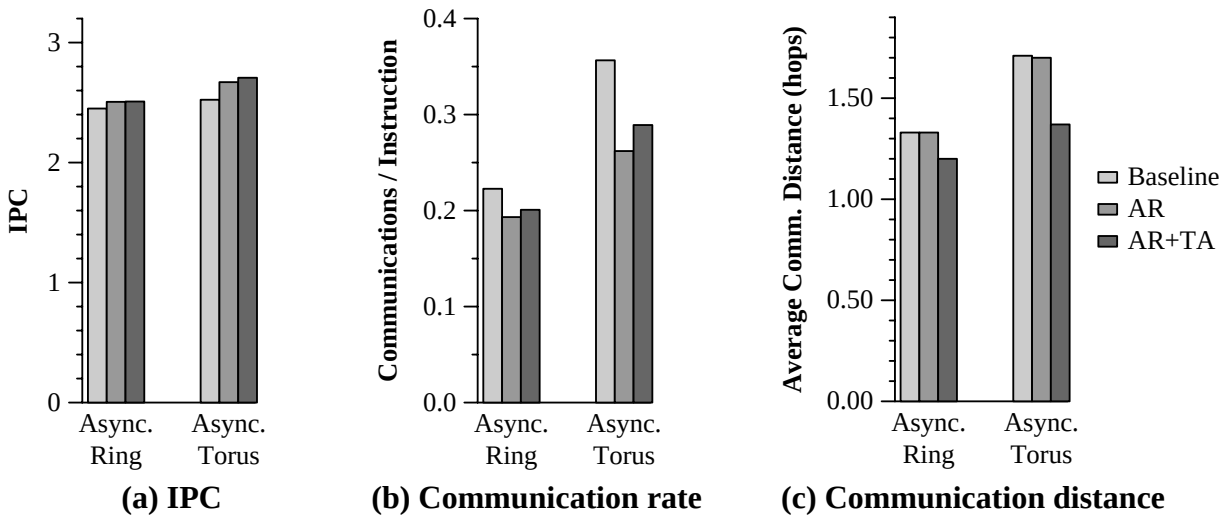


Figure 11. Effectiveness of the AR and TA steering schemes, for a four-cluster ring and an eight-cluster torus

The opportunity to reduce the communication distance occurs only when an instruction has two register operands, and both are available, and they are mapped to two disjoint subsets of clusters. When this happens, at least one communication is required, and the TA technique chooses the cluster that minimizes the longest communication distance to the source operands instead of choosing one of the clusters with a source register mapped. We found that the TA reduces the average communication distance from 1.33 to 1.20 hops for four clusters, and from 1.70 to 1.37 hops for eight clusters, as shown in figure 11c. However, it often occurs that the TA chooses a cluster where none of the operands is mapped, which forces generating a second communication. We found that the TA increases the number of communications per instruction from 0.193 to 0.201 for four clusters, and from 0.262 to 0.289 for eight clusters, as shown in figure 11b. Therefore, the added communication overhead offsets the expected improvements in reducing communication latency.

8.7. Experiments with the SpecInt95

In previous sections, the Mediabench benchmark suite was used for all the experiments. For the sake of higher generality of our conclusions, we run an identical set of experiments with the SpecInt95 benchmark suite.

For 4 clusters, we found that the synchronous ring performs on average 10.2% better than the bus2, while the partially asynchronous ring outperforms bus2 by 12.5%. The performance of the

partially asynchronous ring is very close to that of the ideal ring with unlimited bandwidth (0.3% difference), and it is just 1.4% below that of the ideal crossbar.

For 8 clusters, the average speed-up of the synchronous ring over bus2 is 7.5% whereas the partially asynchronous ring outperforms bus2 by 10.6%. Comparing the partially asynchronous interconnects, the mesh and the torus achieve an IPC 1% and 2.7% higher, respectively than that of the ring. On the other hand, the partially asynchronous torus performance is just 0.5% below that of the ideal torus configuration with unlimited bandwidth, and 3% below that of the ideal crossbar with unlimited bandwidth and 1-cycle latency between any pair of nodes.

Finally we found that the performance of the baseline steering scheme for an 8-cluster torus improves by 2% with the AR technique, and it improves by 2.6% using both the AR and TA techniques. For a 4-cluster asynchronous ring, these techniques showed no significant speedups.

Compared to the results with the Mediabench suite shown in previous sections, these results show similar trends, although in some cases the differences among the various configurations are smaller. However, the same overall conclusions hold for both benchmark suites.

9. Conclusions

In this work we have investigated the design of on-chip interconnection networks for clustered microarchitectures. This new class of interconnects have demands and characteristics different to traditional multiprocessor networks, since low communication latency is essential for high performance whereas achieving a high network throughput is less important. We have shown that simple point-to-point interconnects together with effective steering schemes achieve much better performance than bus-based interconnects. Besides, the former do not require a centralized arbitration to access the transmission medium.

In particular, we have proposed a very simple synchronous ring interconnect that only requires five registers and three multiplexers per cluster and substantially improves the performance of a bus-based scheme. This ring is a good example of the kind of trade-offs that can be done in an on-chip interconnect.

We have also shown that a partially asynchronous ring performs better than the synchronous one at the expense of some additional cost/complexity due to the additional queue required per cluster. However, we have found that a tiny queue will practically never overflow. Thus, instead of using complex flow control protocols, it is much more cost-effective to handle overflows by flushing

the processor pipeline, which is a mechanism that current microprocessors already implement for other purposes (e.g., branch misprediction).

We have explored other synchronous and partially asynchronous interconnects based on meshes and tori, in addition to rings. These three topologies basically differ in their connectivity degree, and consequently, in the average inter-cluster distances. In all the cases, special care has been taken to design simple and fast routing devices, avoiding the need for arbiters and large buffers for in-transit messages at the expense of forbidding a few minimal paths. From our study we extract two main conclusions. First, the interconnects with higher connectivity perform better because they have shorter communication latency. Moreover, point-to-point partially asynchronous interconnects with moderate connectivity/complexity perform close to an idealized crossbar that has every node at one-cycle distance from each other and unlimited bandwidth: a four-cluster ring performs within 2% of the ideal, and an eight-cluster torus performs within 4% of the ideal. Second, despite the low hardware requirements of partially asynchronous interconnects, they achieve a performance close (within 1%) to an equivalent idealized interconnect with unlimited bandwidth and unlimited number of write ports to the register files.

To conclude, the choice of an effective interconnection network architecture together with an efficient steering scheme is a key to high performance in clustered microarchitectures. The simple implementations of point-to-point interconnects that are proposed in this paper are quite effective and scalable.

Acknowledgements

We thank the anonymous referees for their valuable comments. This work is supported by the Spanish Ministry of Education (TIC2001/0995), and by the Generalitat Valenciana (GV04B/487). The research conducted in this paper has been developed using the resources of the CEPBA.

References

[1] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D. Burger, “Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures”, *Proc. 27th Ann. Int’l. Symp on Computer Architecture*, June 2000, pp. 248-259.

[2] A. Aggarwal and M. Franklin, “An Empirical Study of the Scalability Aspects of Instruction Distribution Algorithms for Clustered Processors”, *Proc. Int’l. Symp. on Performance Analysis of Systems and Software*, pp. 172-179, November 2001.

[3] A. Aggarwal and M. Franklin. “Hierarchical Interconnects for On-chip Clustering”, *Proc. Int’l. Parallel and Distributed Processing Symposium*, pp. 63-70, April 2002.

[4] R. Balasubramonian, S. Dwarkadas and D. Albonesi. “Dynamically Managing the Communication-Parallelism Trade-off in Future Clustered Processors”. *Proc. of the 30th. Ann. Int’l. Symp. on Computer Architecture*, pp. 275-286, June 2003.

[5] A. Baniasadi, and A. Moshovos, “Instruction Distribution Heuristics for Quad-Cluster, Dynamically-Scheduled, Superscalar Processors”, *Proc. 33rd. Int’l. Symp. on Microarchitecture (MICRO-33)*, Dec. 2000, pp. 337-347.

[6] M.T. Bohr, “Interconnect Scaling - The Real Limiter to High Performance ULSI”, *Proc. 1995 IEEE Int’l. Electron Devices Meeting*, 1995, pp. 241-244.

[7] D. Burger, T.M. Austin, and S. Bennett, *Evaluating Future Microprocessors: The SimpleScalar Tool Set*, tech. report CS-TR-96-1308, Univ. Wisconsin-Madison, 1996.

[8] R. Canal, J-M. Parcerisa, and A. González, “A Cost-Effective Clustered Architecture”, *Proc. Int’l. Conf. on Parallel Architectures and Compilation Techniques (PACT 99)*, Newport Beach, CA, Oct. 1999, pp. 160-168.

[9] R. Canal, J-M. Parcerisa, A. González, “Dynamic Cluster Assignment Mechanisms”, *Proc. 6th. Int’l. Symp.on High-Performance Computer Architecture*,Jan.2000, pp.132-142

[10] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks, An Engineering Approach*, Morgan-Kauffman, 2003.

[11] K.I. Farkas, P. Chow, N.P. Jouppi, and Z. Vranesic, “The Multicluster Architecture: Reducing Cycle Time through Partitioning”, *Proc. 30th. Int’l. Symp. on Microarchitecture*, Dec. 1997, pp. 149-159.

[12] M. Franklin, *The Multiscalar Architecture*, Ph.D. thesis, C.S. Dept., Univ. of Wisconsin-Madison, 1993.

[13] L. Gwennap, “Digital 21264 Sets New Standard”, *Microprocessor Report*, 10 (14), Oct. 1996.

[14] R. Ho, K.W. Mai, M.A. Horowitz, “The Future of Wires”, *Proceedings of the IEEE*, 89(4): 490-504, Apr. 2001.

[15] G.A. Kemp and M. Franklin, “PEWs: A Decentralized Dynamic Scheduler for ILP Processing”, *Proc. Int’l. Conf. on Parallel Processing*, Aug. 1996, pp. 239-246.

[16] K. Krewell, *Intel Embraces Multithreading*, Microprocessor Report, Sept. 2001, pp. 1-2.

[17] *The International Technology Roadmap for Semiconductors*. Semiconductor Industry Association. 1999.

[18] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, “Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems”, *Proc. Int’l. Symp. on Microarchitecture* (*MICRO-30*), Dec. 1997, pp. 330-335.

[19] D. Matzke, “Will Physical Scalability Sabotage Performance Gains?”, *IEEE Computer 30(9)*: 37-39, Sep. 1997.

[20] R. Nagarajan, K. Sankaralingam, D. Burger and S.W. Keckler, “A Design Space Evaluation of Grid Processor Architectures”, *Proc. Int’l Symp. on Microarchitecture (MICRO-34)*, pp. 40-51, 2001

[21] S. Palacharla, N.P. Jouppi, and J.E. Smith, “Complexity-Effective Superscalar Processors”, *Proc. 24th. Int’l. Symp. on Computer Architecture*, June 1997, pp. 206-218.

[22] S.Palacharla, “Complexity-Effective Superscalar Processors”, Ph.D. thesis, Univ. of Wisconsin-Madison, 1998.

[23] J.-M. Parcerisa, “Design of Clustered Superscalar Microarchitectures”, Ph.D. thesis, Univ. Politècnica de Catalunya, available at http://people.ac.upc.es/jmanel/papers/parcerisa-phdthesis.pdf, April 2004

[24] J.-M. Parcerisa and A. González, “Reducing Wire Delay Penalty through Value Prediction”, *Proc. 33rd. Int’l. Symp. on Microarchitecture (MICRO-33)*, pp. 317-326, Dec. 2000.

[25] J.-M. Parcerisa, A. González, and J.E. Smith, “A Clustered Front-End for Superscalar Processors”, Tech. Report #UPC-DAC-2002-29, Computer Architecture Dept., Univ. Politècnica de Catalunya, Spain, July 2002.

[26] J.-M. Parcerisa, J. Sahuquillo, A. González, and J. Duato, “Efficient Interconnects for Clustered Microarchitectures”, *Proc. 11th Int’l. Conf. on Parallel Architectures and Compilation Techniques*, pp. 291-300, September 2002.

[27] L.-S. Peh and W.J. Dally, “A Delay Model and Speculative Architecture for Pipelined Routers”, *Proc. 7th. Int’l. Symp. on High-Performance Computer Architecture*, pp. 255-266, Jan. 2001.

[28] N. Ranganathan and M. Franklin, “An Empirical Study of Decentralized ILP Execution Models”, *Proc. 8th. Int’l. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998, pp. 272-281.

[29] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J.E. Smith, “Trace Processors”, *Proc. 30th. Int’l. Symp. on Microarchitecture (MICRO-30)*, Dec. 1997, pp. 138-148.

[30] K. Sankaralingam, V.A. Singh, S.W. Keckler, and D. Burger, “Routed Inter-ALU Networks for ILP Scalability and Performance”, *Proc. 21st Int’l. Conf. on Computer Design*, pp. 170-177, October 2003.

[31] M.B.Taylor, W.Lee, S.Amarasinghe, and A.Agarwal, “Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures”, *Proc. 9th. Int’l. Symp. on High-Performance Computer Architecture*, pp. 341-353, Feb. 2003.

[32] J.M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy, *POWER4 System Microarchitecture*, Technical white paper, IBM server group web site, Oct. 2001

[33] A. Terechko, E.L. Thenaff, M. Garg, J.van Eijndhoven, and H. Corporaal, “Inter-cluster Communication Models for Clustered VLIW Processors”, *Proc. 9th. Int’l. Symp. on High-Performance Computer Architecture*, pp. 354-364, February 2003.

[34] M. Tremblay, J. Chan, S. Chaundrhy, A.W. Conigliaro, S.S. Tse, “The MAJC Architecture: A Synthesis of Parallelism and Scalability”, *IEEE Micro* 20(6): 12-25, Nov./Dec. 2000.

[35] J-Y. Tsai and P-C. Yew, “The Superthreaded Architecture: Thread Pipelining with Run-Time Data Dependence Checking and Control Speculation”, *Proc. Int’l. Conf. on Parallel Architectures and Compilation Techniques*, Oct. 1996, pp. 35-46.

[36] H.Wang, L-S.Peh, and S.Malik, “Power-driven Design of Router Microarchitectures in On-chip Networks”, *Proc. 36th. Int’l. Symp. on Microarchitecture (MICRO-36)*, pp. 105-116, Dec. 2003.

[37] K.C. Yeager, “The MIPS R10000 Superscalar Microprocessor”, *IEEE Micro*, 16(2): 28-41, Apr. 1996.

[38] V. Zyuban. *Inherently Lower-Power High-Performance Superscalar Architectures*, Ph.D. thesis, Univ. of Notre Dame, Jan. 2000.

Joan-Manuel Parcerisa received his M.S. and Ph.D. degrees in Computer Science from the Universitat Politècnica de Catalunya (UPC), in Barcelona, Spain, in 1993 and 2004 respectively. Since 1994 he is a full time assistant professor at the Computer Architecture Department at the Universitat Politècnica de Catalunya. His current research topics include clustered microarchitectures, multithreading, value prediction, and cache memory.

Julio Sahuquillo received his BS, MS, and PhD degrees in Computer Science from the Universitat Politècnica de València (UPV), in València, Spain. Since 2002 he is an associate professor at the Computer Engineering Departament at the Universitat Politècnica de València. His current research topics include clustered microarchitectures, multiprocessor systems, cache management, and instruction-level parallelism.

Antonio González received his M.S. and Ph.D. degrees from the Universitat Politècnica de Catalunya (UPC), in Barcelona, Spain. He has been a faculty member of the Computer Architecture Department of UPC since 1986 and he is currently a professor at this department. Antonio leads the Intel-UPC Barcelona Research Center, whose research focuses on new microarchitecture paradigms and code generation techniques for future microprocessors.

His research has focused on computer architecture, compilers and parallel processing, with a special emphasis on processor microarchitecture and code generation. He has published over 150 papers in the areas Power-Aware Microarchitectures; Clustered Microarchitectures; Speculative Multithreaded Processors; Data Value and Data Dependence Speculation and Reuse; Cache Architectures; Register File Architecture; Modulo Scheduling; Code Analysis and Optimization; Mapping Parallel Algorithms to Multicomputers; Prolog-Oriented Architectures; Instruction Fetching Mechanisms; Digital Image Processing.

Dr. González is an Associate Editor of the IEEE Transactions on Parallel and Distributed Systems, ACM Transactions on Architecture and Code Optimization, and Journal of Embedded Computing. He has served on over 50 program committees for international symposia in the field of computer architecture, including ISCA, MICRO, HPCA, PACT, ICS, ICCD, ISPASS, CASES and IPDPS. He has been program co-chair for ICS 2003, ISPASS 2003 and MICRO 2004.

José Duato received the MS and PhD degrees in electrical engineering from the Technical University of Valencia, Spain, in 1981 and 1985, respectively. He is currently a professor in the Department of Computer Engineering (DISCA) at the same university and an adjunct professor in the Department of Computer and Information Science, Ohio State University. He is currently researching multiprocessor systems, networks of workstations, interconnection networks, and multimedia systems. His theory on deadlockfree adaptive routing has been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E router, and the router embedded in the new Alpha 21364 microprocessor. He coauthored Interconnection Networks: An Engineering Approch with S. Yalamanchili and L.M. Ni (IEEE CS Press). Dr. Duato served as an associate editor of the IEEE Transactions on Parallel and Distributed Systems from 1995 to 1997. He is currently serving as an associate editor of the IEEE Transactions on Computers. He has been or is a member of the program committee for several major conferences (ICPADS, ICDCS, Europar, HPCA, ICPP, MPPOI, HiPC, PDCS, ISCA, IPPS/SPDP, ISPAN). He served as general cochair for the International Conference on Parallel Processing 2001. He is a member of the IEEE.