Efficient Interconnects 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
{jmanuel,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 microarchitectures. This new class of interconnects has different demands and characteristics than traditional multiprocessor networks. In a clustered microarchitecture, a low inter-cluster communication latency is essential for high performance.

We propose point-to-point interconnects together with an effective latency-aware instruction steering scheme and show that they achieve much better performance than bus-based interconnects. The results show that the connectivity of the network together with latency-aware 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.

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 [13, 24], 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 [17]. On the other hand, projections about future technology trends foresee that long wire delays will scale much slower than gate delays [1, 3, 11, 14, 16]. 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 [8, 9, 10, 17, 19, 23, 24, 27]. 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 [5, 19]. Many steering heuristics have been studied to reduce the communication rate [2, 6], and value prediction has been proposed to hide the communication latency [19]. The alternative approach proposed in this work consists of reducing the communication latency, by designing networks that reduce the contention delays and proposing a topology-aware instruction steering scheme that minimizes communication distances. 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 completely 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 microarchitectures have assumed interconnection networks that are either an ideal-

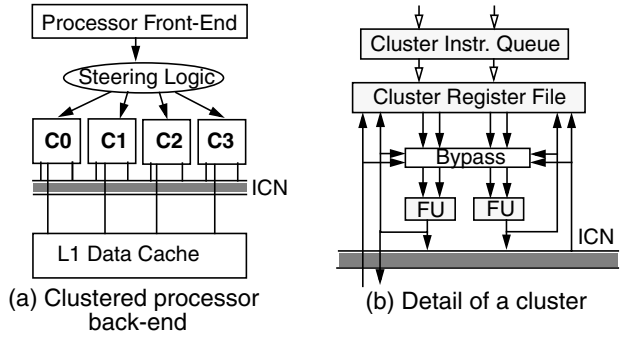

Figure 1. Clustered microarchitecture

ized model ignoring complexity issues [2, 19], or they consider only 2 clusters (Multicluster [8], Alpha 21264 [10]), or they assume a simple but long-latency ring [12]. In this paper, we explore several alternative interconnection networks with the goal of minimizing latency while keeping the cluster complexity low. To the best of our knowledge no other work has addressed this issue. 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, i.e., just a single dedicated write port in each register file, and achieve performance close to those of idealized models without contention.

The rest of this paper is organized as follows. Section 2 gives an overview of the assumed clustered microarchitecture. Section 3 describes the proposed interconnection networks, and Section 4 analyzes the experimental results. Finally, Section 5 summarizes the main conclusions of this work.

2. Microarchitecture Overview

Several microarchitectures with different code partitioning strategies have been proposed [21]. They partition the code either at branch boundaries [9, 22, 25], or grouping dependent instructions together [5, 6, 8, 12, 17, 18]. The microarchitecture assumed in this paper is based on a dependence-based paradigm [6, 19]. It differs from others in that the register file is distributed, and 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 [26] or the Alpha 21264 [10] 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 are shown in Table 1.

Table 1. Default machine parameters

Parameter	Configuration
I-cache L1	64KB, 32-byte line, 2-way assoc, 1cycle hit
Branch Predictor	Hybrid gshare/bimodal: Gshare has 14-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)	1, 2, 4, 8
Phys. regs. per C	56 int + 56 fp
IQ size per C	16
Issue width per C	2
Fetch/Decode width	8
F.U. per C	2 int ALU,1 int mul/div,1 fp ALU,1 fp mul
ROB size	128
LSQ size	64
Issue	Out-of-order issue. Loads may issue when prior address stores are known
D-cache L1	64KB, 32 byte line, 2way set-associative, 3 cycle hit time, 3R/W ports
I/D-cache L2	256KB, 64 byte line, 4way assoc, 6 cycle hit

connects, we assumed a simple centralized front-end and data cache, although some strategies are currently being investigated to distribute these components as well [20]. 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 proposed techniques in this paper can easily be generalized for heterogeneous clusters.

2.1. 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, 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 of the

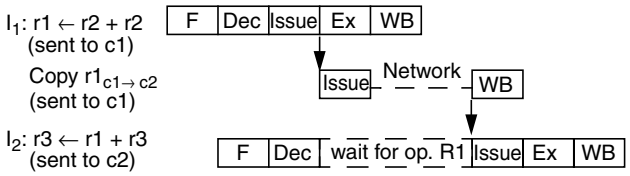

Figure 2. Sample timing of a communication between two dependent instructions I_1 and I_2, steered to clusters c1 and c2 respectively (the arrows mean wakeup signals, and communication latency is 2 cycles).

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.

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.

2.2. A Topology-Aware Steering Heuristic

Our instruction steering heuristic is a variation of the baseline heuristic proposed by Parcerisa and González [19] for a similar microarchitecture. The baseline steering scheme works in the following way: if the workload of the clusters is not much imbalanced, the algorithm first follows a primary criterion which selects the clusters that minimize communication penalties, and next it follows a secondary criterion which chooses the least loaded of the above selected clusters. However, when the workload imbalance exceeds a given threshold, the primary criterion does not apply (refer to [19] for further details about the workload metrics). To satisfy the primary criterion, i.e. to minimize communication penalties, the heuristic distinguishes two cases: (i) if all the source operands are available, it chooses the clusters that have the highest number of source operands mapped, and (ii) if any operand is not available, it chooses the cluster where it is going to be produced.

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 steering heuristic that is aware of the network topology 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, in such a way that when all source operands are available (case (i)), it 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 enhanced heuristic will select only clusters 1 and 2.

3. The Interconnection Network

Each copy instruction sends a message through the network, including the copied operand, and the corresponding register tag. In this section we discuss several design trade-offs and constraints regarding the design of the interconnection network, and next we describe in detail those that have been experimentally analyzed for architectures with 4 and 8 clusters.

3.1. Routing Algorithms

Interconnection networks have been widely studied in the literature for different computer areas such as multicomputers and network of workstations (NOWs) [7]. In these contexts, communication latencies are thousands of processor cycles long, and routing decisions take several cycles. 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 5, and also by other previous works [2, 5]. Thus, in this context 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 taken 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.

3.2. Register File Write Ports

The network is connected to 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, which increases the wakeup delay by a quadratic factor with respect to the number of tags [17]. 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 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 (0.22 communications per instruction for 4 clusters [19]), 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.

3.3. 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. Clusters at a one cycle distance are considered neighbors. Second, only neighbor clusters 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).

3.4. 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 3). 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

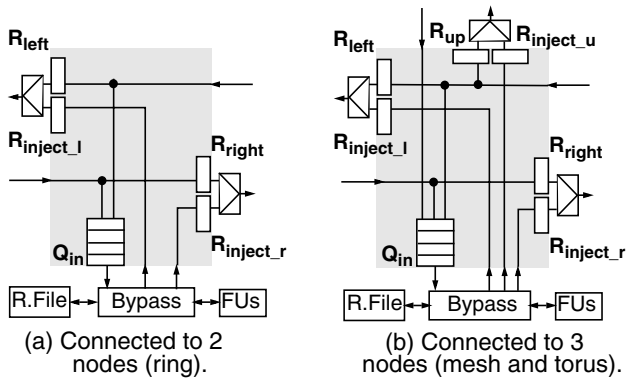

Figure 3. Router schemes for asynchronous point-to-point interconnects

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 messages compete for the same output link. This conflict does not exist in a ring (Figure 3a) but may happen if a node has more than 2 neighbors, like in a mesh or a torus, where each router may have up to 4 links. Note, however, that since we have considered only small meshes with 4 and 8 clusters, each router has never more than 3 links (Figure 3b).

A *copy* instruction is kept in the issue buffer until both its source operand is available and it secures the required bus or link, so no other buffering storage is required. That is, the scheduler handles the router injection registers (R_{inject}, in Figure 3) as any other resource. However, while access requests for a bus-based network are sent to a distant centralized arbiter, the arbitration for a point-to-point network is done locally at the source cluster by simply monitoring in-transit messages at the router stage registers (R_{right}, R_{left} and R_{up}). Eventually, the *copy* is issued and the outgoing message stays in one of the R_{inject} output registers while it is being transmitted.

The router also interfaces with the cluster datapath. For partially asynchronous networks, the router includes an input FIFO buffer (Q_{in}, in Figure 3) 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. For synchronous networks, the router is still less complex. By appropriately scheduling the injection of messages at the source cluster (more details are given later), the proposed scheme guarantees that there is no more than one input message per cycle. Therefore, the router requires just a single register (not shown), instead of the FIFO buffer.

3.5. Bus versus Point-to-Point Interconnects

Although our analysis mainly focuses on point-to-point networks, we also study a bus interconnection, for comparison purposes. It is made up of as many buses as

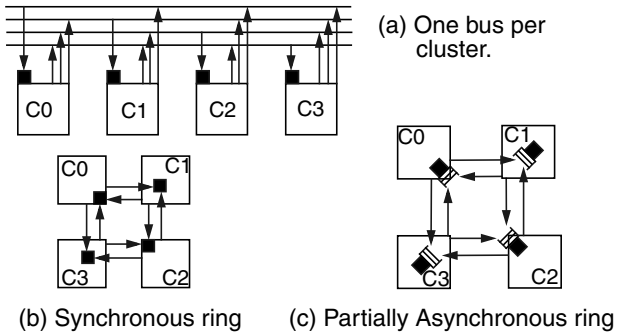

Figure 4. 4-cluster topologies

clusters, each bus being connected to a write port in one cluster, and each cluster being able to send data to any bus (Figure 4a). 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 the clusters, thus leading to long wires and long transmission times, which can drastically reduce the bandwidth if the bus transmission time is longer than one cycle[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 requiring shorter wires and having smaller parasitic capacitance (less devices attached to a point-to-point link than to a bus). Fourth, network cost is lower than for a configuration with as many buses as 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.

3.6. Four-Cluster 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 4). We also compare their performance to that of an idealized ring, which represents an upper bound for ring networks. In a 4-cluster point-to-point topology, inter-cluster distances are either one cycle (messages sent among adjacent nodes) or two cycles (messages sent among non-adjacent nodes). Below we describe these topologies.

Bus2. This is a realistic bus model 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 4a), and a very simple centralized bus arbiter. 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.

Synchronous Ring. This topology 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 issue logic schedules 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. During odd cycles, a source cluster *src* is allowed to send a short-distance message (D=1) to its adjacent cluster in the clockwise direction (*(src +1) mod 4*), and a long-distance 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 3.4), a copy instruction may have to wait until both the requested link is available and the cycle parity is appropriate.

Table 2. Rules to secure a link in the source cluster *src* (D refers to distance in cycles)

Direction	Odd Cycle		Even Cycle	
	D	Target Cluster	D	Target 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 [7], 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.

Partially Asynchronous Ring. Typical asynchronous networks include buffers both in the intermediate routers, to

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

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) [7]. The former are removed in our design, like in the synchronous ring. However, we still need the latter, since two messages can 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 4.4. 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.

Ideal Ring. For comparison purposes, we consider an idealized ring whose inter-cluster distances are the same as those of the realistic ring (as previously discussed), 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 write ports for incoming messages in each cluster). The performance of this ideal ring is an upper bound for the performance of any ring that has the same link latencies as the ones assumed in this study.

3.7. Eight-Cluster 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 5 shows these two new schemes. Below, we describe each scheme in detail.

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).

Synchronous Ring. In this section, the routing scheme discussed above for a 4-cluster synchronous ring is extrapolated to 8 clusters. Like in the 4-cluster configuration, at issue time the scheduler of copy instructions (shown in Table 3) must ensure that only one message arrives at a time at a given cluster, because there is only one write port. Note that distances in an 8-cluster ring topology range from 1 (D=1) to 4 (D=4) cycles.

Table 3. Rules to secure a link in the source cluster *src* (D refers to distance in cycles)

Direction	Odd Cycle		Even Cycle	
	D	Target Cluster	D	Target 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*

Mesh. A mesh topology (see Figure 5a) 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 router input link could compete to access the same output link.

Our approach is to avoid buffers for storing in-transit messages by simply constraining the connectivity of some paths. More precisely, a message flowing through one of the dashed links in Figure 5a must have the node at the link's end as its destination, and it must come from either the node at the link's origin or from a predetermined associated link among those to which it is connected. For instance, if one message is sent from cluster C2 to C7, it must be routed through C6. It cannot be sent through C3 because the link C2-C3 does not end at the message destination. Also, a message from C0 to C3 must be routed through C1 because the link C0-C2 is not associated to the link C2-C3.

Again, deadlocks are avoided by using the same clock signal for all the routers and transmitting messages synchronously.

Torus. A torus has smaller average distance than a mesh (see Figure 5b). In some nodes (C0, C1, C4 and C5) more than one input link 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 3b).

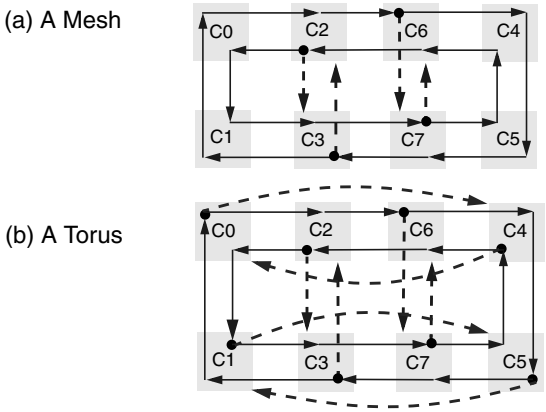

Figure 5. Additional topologies for 8 clusters. A black dot at the end of a link means that there is more than one link that can be followed for the next hop if the corresponding node is not the destination. Messages can always be routed through solid links but dashed links represent the end of the corresponding path; thus, they are only used when they are the final ones.

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 2-hop route from C4 to C1. However, due to the poor utilization of the network (as we will show later) this is a minor drawback.

Again, deadlocks are avoided as indicated above. On another issue, when mapping torus links on silicon, some links may be longer than the shorter ones (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.

Ideal Torus. For comparison purposes, we also consider an idealized torus model, with distances 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 write ports connected to the network in each cluster. The performance of this model is an upper bound on the performance of the realistic torus.

4. Experimental Results

In this section it is described the simulation environment and it is evaluated the different network architectures proposed above.

4.1. Experimental Framework

To perform our microarchitectural timing simulations, we have extended the sim-outorder simulator of the SimpleScalar v3.0 tool set [4] with all the architectural features described above, including the different interconnection network topologies.

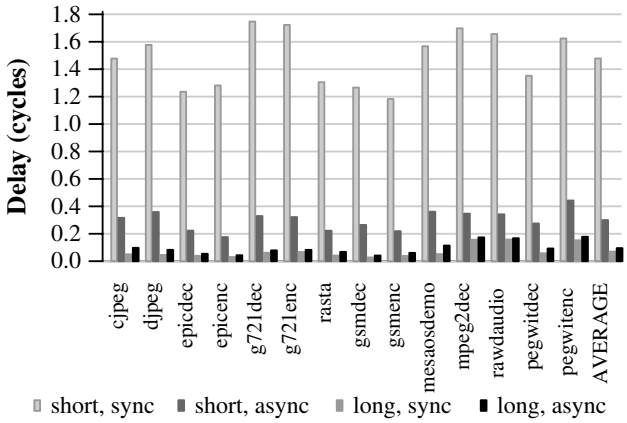

Figure 6. Average contention delays per communication, with ring interconnects.

We have selected a subset of 14 benchmarks, those achieving higher IPC, from the Mediabench benchmark suite [15]. 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.

All topologies maintain the same processor model. Table 1 summarizes the machine parameters used through the simulations.

4.2. 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.

As shown in Figure 6, short-distance messages wait for longer than long-distance ones. 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 is the same for both ring interconnects, it does not explain the differences observed between the two rings.

Figure 6 shows that the contention delay of short-distance messages for a synchronous ring (1.48 cycles) is about 4.9 times longer than for a partially asynchronous one (0.30 cycles). In contrast, the contention caused to long-distance messages for a synchronous ring (0.07 cycles) is just slightly lower than for a partially asynchronous one (0.10 cycles). These differences are due to the different ways each interconnect avoids conflicts between messages

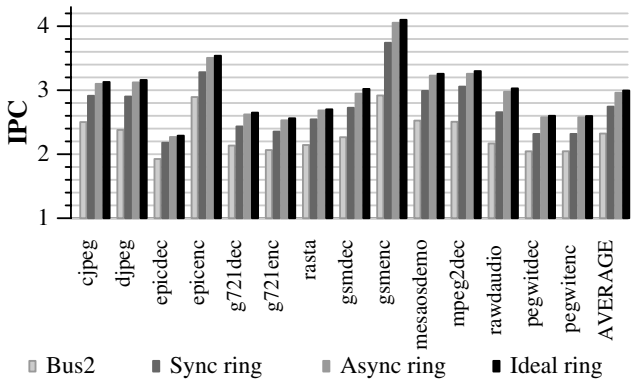

Figure 7: Comparing 4-cluster topologies

that require access to the same write port: in a synchronous ring, these conflicts are prevented by ensuring that a short-distance 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 long-distance message from C1 to C3 (see Figure 4b) 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 short-distance messages will be delayed if there are in-transit long-distance 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 in the destination cluster router until it gains access to the register write port.

To summarize, the long delays caused to short-distance messages by the scheduling constraints of the synchronous ring make its overall contention delay higher than for a partially asynchronous one. In addition, since the steering algorithm tries to minimize the distance, near 80% of the messages are short-distance, and therefore these messages have a higher impact on the overall contention delay. As a consequence, the partially asynchronous ring performs better than the synchronous one, as shown below.

4.3. Performance of Four-Cluster Interconnects

Figure 7 compares the performance, reported as number of committed instructions per cycle (IPC), of a 4-cluster processor for the different proposed interconnects.

The two ring interconnects consistently achieve better performance than the bus topology, for all benchmarks. This is mainly because point-to-point networks achieve shorter latency since the steering algorithm tries 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 [19] (e.g. it is on average 0.18 communications per instruction, on a partially asynchronous ring). The IPC achieved by the synchronous ring is, on average, 18.1% higher than that achieved by the bus, while the partially asynchronous ring performance is 27.4% higher than that of the bus.

The performance of the partially asynchronous ring is very close to that of the ideal ring (just 1.1% difference), which shows that increasing the number of links or the number of register 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 a single link between adjacent clusters and a single register write port for incoming messages is clearly the most cost-effective design.

4.4. Queue Length

Typical partially 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 96% of the messages do not have to wait because they find the queue empty, and the maximum number of occupied entries was 9. 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	1327534	96.20	96.20
1	47136	3.42	99.61
2	4807	0.35	99.96
3	484	0.04	100.00
4	26	0.00	100.00
5	1	0.00	100.00
>= 6	0	0.00	100.00

Although 9-entry queues are long enough in our experiments, the model should ensure that data is never lost, 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

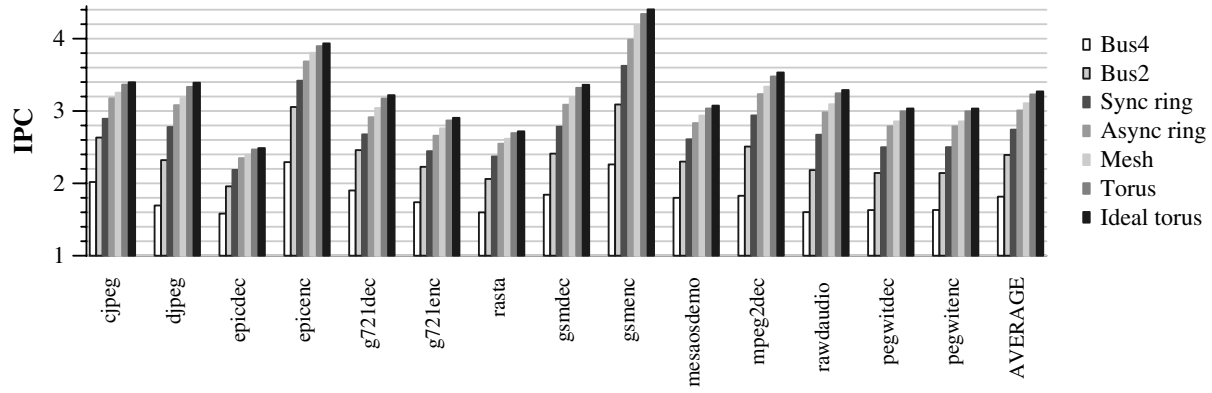

Figure 8: Comparing 8-cluster topologies

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 is to flush the pipeline in case of an overflow, very much like in the case of a branch misprediction, and to restart again execution at the instruction that generated the message that caused the overflow. This approach requires minimal additional hardware and it produces negligible performance penalties (for 9-entry queues there is no penalty at all for our benchmarks).

4.5. Performance of Eight-Cluster Interconnects

In this section, we evaluate the 8-cluster network interconnects described in Section 3.7, for a 16-way issue architecture. Figure 8 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 14.6% whereas the partially asynchronous ring outperforms bus2 by 25.7%.

Comparing the partially asynchronous topologies, the mesh achieves an IPC 3.3% higher than that of the ring, while the IPC of the torus is 7.3% 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 (just 1.2% difference).

4.6. Performance of the Topology-Aware Steering

In all the previous experiments it was assumed the topology-aware instruction steering scheme described in Section 2.2. This scheme differs from the previously proposed baseline scheme, because it minimizes communication distances (hence latencies) for point-to-point interconnects. Figure 9 compares both steering schemes for two configurations, one with 4 clusters and an asynchronous ring, the other with 8 clusters and an asynchronous torus. It shows that the topology-aware steering scheme increases IPC by 2.5% with 4 clusters, and by 16.5% with 8 clusters.

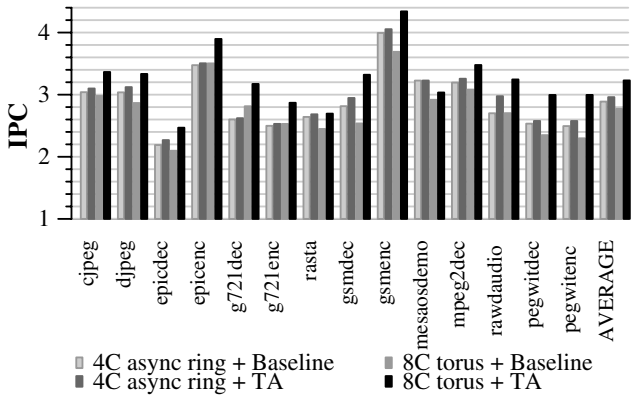

Figure 9: Baseline vs. topology-aware (TA) steering schemes, for 4 and 8 clusters

The performance improves because the average communication latency is reduced.

5. Conclusions

In this work we have investigated the design of on-chip interconnection networks for clustered microarchitectures. This new class of interconnects has different demands and characteristics than traditional multiprocessor networks, since a low communication latency is essential for high performance. We have shown that simple point-to-point interconnects together with effective latency-aware steering schemes achieve much better performance than bus-based interconnects. Besides, the former does not require a distant 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.

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 (i.e., branch misprediction).

We have explored other synchronous and partially asynchronous interconnects such as meshes and a torus, in addition to rings. These three topologies basically differ in their connectivity degree, and consequently, in the average inter-cluster distances. From our study we extract two main conclusions: (i) higher connectivity results in significant performance improvements, and (ii) despite the low hardware requirements of the proposed partially asynchronous interconnect, it achieves a performance close to an equivalent idealized interconnect with unlimited bandwidth and number of ports to the register file.

To conclude, the choice of an effective interconnection network architecture together with an efficient latency-aware steering scheme is a key to high performance in clustered microarchitectures. Simple implementations of point-to-point interconnects 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 and by the FEDER funds of the European Union, under contract CYCIT TIC2001-0995-C02-01.

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. 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.

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

[4] 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.

[5] 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.

[6] 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

[7] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks, An Engineering Approach,* IEEE Computer Society Press, 1997.

[8] 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.

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

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

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

[12] 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.

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

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

[15] 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.

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

[17] 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.

[18] S.Palacharla, "Complexity-Effective Superscalar Processors", Ph.D. thesis, Univ. of Wisconsin-Madison, 1998.

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

[20] J.-M. Parcerisa, A. González, and J.E. Smith, "Building Fully Distributed Microarchitectures with Processor Slices", tech. report UPC-DAC-2001-33, Computer Arch. Dept., Univ. Politècnica de Catalunya, Spain, Nov. 2001.

[21] 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.

[22] 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.

[23] 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

[24] 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.

[25] 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*, 1996, pp. 35-46.

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

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