Low-level Router Design and its Impact on Supercomputer System Performance *

V. Puente, J.A. Gregorio, C. Izu[1], R. Beivide and F. Vallejo
University of Cantabria
39005 Santander, Spain
email: {vpuente,jagm,mon,fernando}@atc.unican.es
[1]University of Adelaide
SA 5005, Australia
email: cruz@cs.adelaide.edu.au

Abstract

Supercomputer performance is highly dependent on its interconnection subsystem design. In this paper we study how different architectural approaches for router design impact into system performance when running real parallel applications. A thorough methodology has been employed to quantify this impact. Architectural router decisions have been chosen taking into account the constraints of the underlying VLSI technology. After that, an exhaustive evaluation of the interconnection network under standard synthetic traffic has been carried out. Finally, an execution-driven simulation environment has been used to assess the consequences of several router designs on the performance of the entire machine. We will show that low-level decisions, as the adequate selection of router's arbiter, significantly reduce the execution time of parallel applications. To illustrate the effects of the router architecture on system performance two benchmarks were selected: Radix and MP3D.

1 Introduction

In the field of high-performance computing, distributed shared-memory multiprocessors (DSMs) are becoming widespread. These parallel computers implement a single address space, either with coherent caches (SGI Origin 2000 [13]) or without them (Cray T3E [18]). The communication time involved on fetching remote data is one of the main overheads which limits the performance of many parallel applications. Moreover, cc-NUMA machines impose additional overheads due to synchronization amongst processes and coherence maintenance. As processor computing power increases, communication performance should increase accordingly in order to adequately balance the system.

*This work is supported in part by TIC98-1162-C02-01

Last decade's network improvements were based on the use of pipelined switching techniques, wormhole or virtual cut-through, which considerably reduce message latency by routing a message as soon as its header reaches the router [6]. These techniques make latency less sensitive to the distance in the network provided that messages are long enough, facilitating the search for optimal topologies. Several researchers recommended the use of low-dimensional direct networks in the k-ary n-cube class [1, 7]. As a result, the use of wormhole-based bidimensional or three-dimensional meshes and tori or limited-degree hypercubes is overwhelmingly dominant in multicomputers and DSMs [3, 11, 18].

The interconnection network should minimize message latency, so routers must have a intra-node delay as low as possible. Consequently, dimensional order routing (DOR) has been a common option due to its simple hardware implementation [22]. However, DOR networks exhibit poor behavior for medium to high loads because of increasing link contention. Thus, the latest routers use mechanisms such as larger buffer capacity, adaptive routing and virtual channels in order to alleviate contention. The Cray T3E router uses these three techniques [18]. Other routers such as the Intel Cavallino[3], the SGI SPIDER [11], and the MIT Reliable Router [9] also employ a few virtual channels per physical link with the latter implementing fully adaptive routing as well. These features improve throughput significantly [8, 10], and may reduce the execution time for bandwidth-limited parallel applications. However, virtual channels and adaptive routing have been shown to increment router delay [5].

Recent work on the communication demands of parallel application has shown that many applications are latency sensitive, so it has been suggested that routers should implement neither virtual channels nor adaptive routing [24].

In [20] we proposed an adaptive virtual cut-through router with lower clock cycle and node delay than its oblivious wormhole counterpart. This was achieved by considering the VLSI constraints during the architectural design of the router. A critical element of any adaptive router is the arbiter, which role is to match the input requests with the available output ports. This is

a simple task in a DOR router as each input requests only one of the outputs and, therefore, each output port assignment is independent of the rest. Complexity rises with adaptivity because each incoming message may request one or *more* output ports. Besides, adaptive routers add two or more virtual channels in order to avoid message deadlock. This, at least, duplicates the number of possible requests. Independent arbitration for each output port may lead to multiple output resources assigned to the same message. A centralized arbiter can accept multiple input requests from each incoming message, matching them to all available output ports. Obviously, this approach allows for the maximum number of packets to be transmitted from input to output [21]. However, it is easy to forecast a high VLSI implementation cost. There are other approaches to reduce its complexity including the following two:

- A distributed arbiter per input channel accepting multiple output requests but servicing only one input channel per cycle. A similar option was used in the Cray T3E router [18].
- A distributed arbiter per output port that forces each input to send a single request per cycle. This is the approach taken in our adaptive router [20].

From the architectural point of view, these two approaches penalize throughput in comparison to the centralized option. To properly evaluate the virtues of a router design we need to consider, not only the network behavior in terms of cycles, but also the VLSI cost that determines the network clock cycle as well as the demands of parallel applications that will make use of that network.

According to this methodology, three arbiters were designed using VHDL tools. We then carried out a thorough evaluation in terms of node performance, network performance and execution time of parallel applications. The differences among these arbiter implementations clearly translate into gains both at the network level and at the application level.

The rest of this paper is organized as follows. Section 2 will present the motivation and framework for studying the router's arbitration of messages as well as their architectural details. Section 3 will provide VLSI cost values for each arbiter and their corresponding impact on node performance. Section 4 extends the evaluation from one node to a given size network and Section 5 studies the impact of network performance in the execution time for two parallel applications. Finally, Section 6 will summarize the findings of this work.

2 Motivation and Related Work

As mentioned above, our interest on arbiter's evaluation originated from the VLSI design of an adaptive virtual cut-through (VCT) router [20]. Thus, we will first present an overview of such a router. This router was specified using the hardware description language VHDL.

2.1 The Adaptive Bubble Router

Internal router components are clocked synchronously but communication with neighbor routers are asynchronous in order to avoid clock skew problems. The main blocks of the proposed router architecture are shown in Figure 1. Each input link has two FIFO queues associated to it, namely *adaptive* queue and *escape* queue. Each of the input queues is conceptually the same as a virtual channel in a wormhole-based context. As we assume a VCT context, each queue must be able to store at least one packet; we measure the capacity of a queue in terms of packet units.

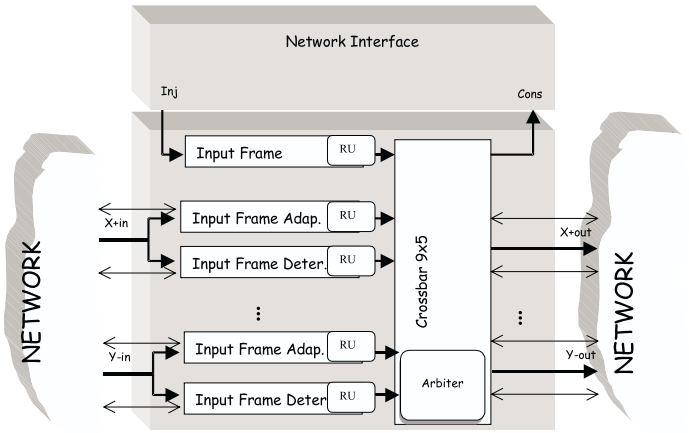

Figure 1: Router organization for a 2-D torus.

Input queues and output ports are connected through a crossbar. As link multiplexing is performed on a packet-by-packet basis, a virtual channel controller is not required at the crossbar output.

Packets can move from an adaptive queue to an escape queue or viceversa, fulfilling the following premises:

- Packets travelling through escape queues use DOR paths and the access to each dimensional ring is regulated by Bubble flow control [4]. This differs from VCT in that packet injection into the ring is only allowed if there are at least **two** empty packet units (Bubble condition) in the set of queues corresponding to the dimension (and direction) requested by the incoming packet. The first free packet unit must be located at the requested queue, and our implementation checks locally the existence of the second free packet unit. This policy guarantees deadlock-freedom in each unidirectional ring.
- Packets travelling through adaptive queues can use any minimal path and are regulated by virtual cut-through flow control.

Packets using a escape queue can freely use, if available, an adaptive queue at the next router. The resulting routing algorithm is minimal fully adaptive and reduces hardware requirements with respect to previous proposals.

A routing unit, attached to each input queue, decodes the packet header flits and determines the list of possible outputs. The selection function examines the possible outputs in a fixed order, which for 2D routers is the following:

1. The adaptive queue of the neighbor along the incoming dimension, if the first offset is not zero.
2. The adaptive queue of the neighbor along the other dimension, if the second offset is not zero.
3. The escape queue of the neighbor along the first dimension, if the x offset is not zero or the escape queue of the neighbor along the second dimension if the x offset is zero. Prior to requesting an escape queue, the routing unit (RU) verifies the Bubble condition.

Thus, messages only turn when they encounter a busy link or they exhaust their path in that dimension. When a request is granted, the RU adjusts the header's phits so that the offset of the granted dimension is decremented, and sent first. Finally, it is worth mentioning that the operations carried out by the routing unit depend on packet destination but they do not depend on the queue storing the packet. Thus, the RU is identical for both adaptive and escape queues, except for the need to check the bubble condition.

The arbitration process in a k-ary n-cube involves $2n+1$ input queues and $n+1$ output ports, that is 9 input queues and 5 output ports in a 2D torus. The arbiter receives requests from each input's RU, performs the matching of requests to outputs, then sends the corresponding signals to the successful input(s) and set the crossbar's datapath. Thus, any variation on the arbiter's design impacts in these other two components and we will describe their interaction for each router alternative.

For example in a 2D router, the complexity of arbitration is given by the maximum number of requests which is 27, that is three simultaneous requests from each of the 9 input queues. Considering both the selection function described above and the status of output ports and their adjacent queues, the arbiter may grant up to 5 of those requests.

2.2 The Output Arbiter Crossbar (OAC)

Our initial approach, called Output Arbiter Crossbar (OAC) limits each routing unit to request a single output port per cycle from its set of possible outputs.

As there is only one request per input, the complexity of the OAC arbiter is similar to that of the DOR's arbiter: (2n+1) input requests directed to any of the (n+1) ports. This unit grants as many requests as possible, arbitrating amongst inputs contending from the same output. The $(2n+1)$x$(n+1)$ crossbar can be subdivided into $(n+1)$ multiplexers of $(2n+1)$ inputs. Thus, arbitration and switching logic is distributed for each output port as shown in Figure 2.

The number of requests per input queue is reduced to one by applying the selection function at the RU. Each cycle, the RU will request one of the possible outputs, in the order given by the selection function, until one of them is granted.

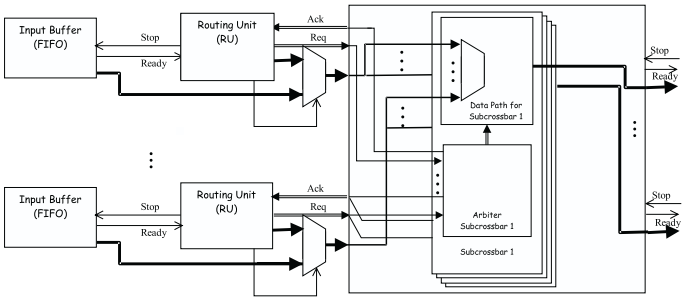

Figure 2: Partial Router Structure for Output Arbiter Crossbar (OAC).

2.3 The Symmetric Arbiter Crossbar (SAC)

Tamir describes a switch in which each input port contends for multiple output ports but needs only one for full utilization [21]. Similarly, each output port contends for multiple input ports and needs one for full utilization. Thus, the arbitration task is symmetrical with respect to inputs and outputs. The arbiter sees all the requests and it takes a combined decision regarding output selection for each input and contention for a given output.

Tamir proposed a Wave Front Arbiter as the one shown in Figure 3, in which each row r_i represents the set of requests from input i, and each column the competing requests for output j. The arbitration cells for each request reached their final configuration in a *wave form* that moves diagonally from the top left to the bottom right corner of the arbiter. Although this scheme achieves good performance [21] is not exempt of fairness and starvation anomalies. Tamir's solution is based on circulating a token which selects the row and column with highest priority. The circulating token is implemented using a circular shift register.

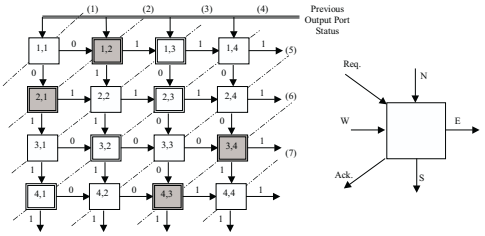

Figure 3: (a)Wave Front Arbiter 4x4 and (b)arbiter's basic cell unit

Tamir's arbiter gives equal priority to all requested outputs. In other words, the selection function has no fixed priorities but the priority (first output examined) rotates as the token circulates. This differs from our router proposal in which messages only turn when blocked in their current dimension. Besides, escape

queues have always low priority, thus they should always be located at the end of the wave front. Our SAC arbiter is based on the WFA but it reorders the output columns in such a way that the selection function is preserved. Thus, the wave's starting cell changes from row to row each cycle, but starts always in the first output column.

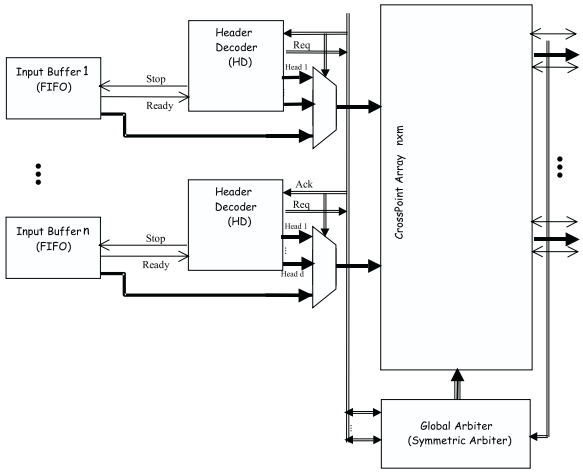

Figure 4: Partial Router Structure for Symmetric Arbiter Crossbar (SAC).

2.4 The Sequential Input Crossbar (SIC)

The third approach is similar to OAC in that simplifies arbitration by reducing the number of processed requests per cycle. In this case, the arbiter services only one input's routing unit (Header Decoder) per cycle. Incoming messages are sequentially serviced in a round-robin fashion. Thus, in each cycle the arbiter only checks the availability of the requested outputs by the current HD and, if two or more are available, it selects the one with higher priority.

If we draw a schematic design of a router using the SIC arbiter it will look alike the router of Figure 4. But internally is much simpler as it only examines the requests from one input at a time, so arbitration logic such as the WFA is replaced by a simple token structure that selects the next input to be serviced.

3 Arbitration and Node Performance

The three arbiter schemes have been designed and incorporated into the Adaptive Bubble Router. This section shows the impact of each scheme in the router features, in terms of VLSI cost and node performance.

3.1 Arbiter and Router Implementation

We have designed the three arbiters using the *Synopsys v1997.08* synthesis tool. Then, we mapped our design into 0.7 μm (two metal layers) technology from ATMEL/ES2 foundry under typical working conditions with standard cells from *Es2 Synopsys design kit V5.2.* Although the obtained results for area and time are estimated by *Synopsys* in a pessimistic way [23], they provide us with values very close to the physical domain.

Three channel widths were considered: 9 (8 data bits + 1 tail bit), 17 and 33 bits. The main impact of the channel width is on router area and pin count but it does not impact on the arbitration complexity or clock cycle. From now on, we will consider that the channel width is 33 bits, that is 4 bytes per phit plus 1 tail bit.

For SIC and SAC, the arbiter unit exhibits the highest delay, imposing the router's clock cycle. OAC arbitration, though, exhibits a much shorter delay than SIC, in spite of accepting more requests, because the selection function is already applied at the Routing Unit. Even after integrating the arbitration and crossbar setup in a single stage, we obtain a through delay similar to that of the RU stage, thus resulting in a well-balanced pipeline. Consequently, the interaction between the routing unit, the arbiter and the crossbar varies for each alternative, resulting in different pipeline organizations. Figures 5 and 6 show the pipelines for OAC and SIC/SAC respectively.

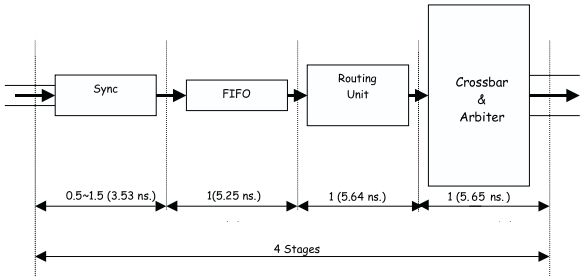

Figure 5: Bubble router pipeline with OAC arbitration (Critical Paths for 2-D Router).

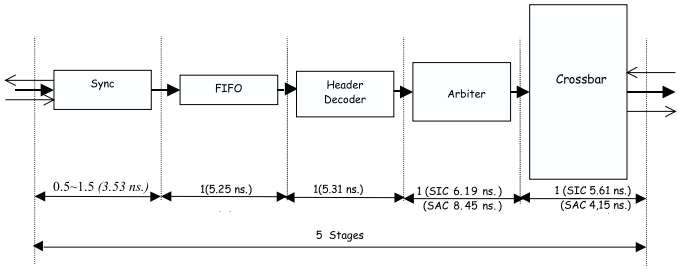

Figure 6: Bubble router pipeline with SIC/SAC arbitration (Critical Paths for 2-D Router).

The cost values for 2D and 3D router nodes are shown in Table 1. As expected, SAC requires the most area and exhibits the highest delay. Both OAC and SIC reduce arbitration time significantly as well as arbiter's area.

3.2 Single Node Performance

Each router performance depends not only on the arbitration time, but also on the ability to match multiple

	2D			3D		
Arbiter Type	**Router Cycle(ns)**	**Arbiter's Area(mm^2)**	**Router's Area(mm^2)**	**Router Cycle(ns)**	**Arbiter's Area(mm^2)**	**Router's Area(mm^2)**
OAC	5.65	14.33	43.72	6.28	24.5	66.83
SAC	8.45	21.68	52.04	11.69	39.14	71.59
SIC	6.19	8	32.5	7.5	16	50.1

Table 1: Area and time values for each arbiter scheme in a 2-D and 3-D router node under typical conditions.

requests to the available outputs. Although the physical implementation would be the best way to assess the performance of a router as part of a network in a parallel system, accurate results can also be obtained by employing simulation tools. We used a register level transfer (RTL) simulator called *SICOSYS* [19], which reflects the implementation cost in terms of pipeline stages and clock cycle. This simulator provides accurate results which are extremely close to those obtained through VHDL simulation [20]. This approach exhibits lower simulation times, thus reducing the design cycle.

All our experiments applied the component delays previously obtained, and set the input queues capacity to 4 packets. The first experiment stressed the node capabilities by loading all its input channels to their maximum rate. To do so, we attached injection and consumption *virtual units* to each input and output link of the router node. A continuous stream of packets was supplied and node throughput recorded. Considering a network with a large radix k, the number of messages delivered to the local node is negligible when compared to the volume of transit traffic. Hence, node throughput is limited to $2 * d * B$, where d represents the number of dimensions and B the link bandwidth. The incoming traffic emulated the behavior of messages in a real network under uniform random traffic, fulfilling the rules dictated by the Adaptive Bubble Routing algorithm.

Figures 7 and 8 shows the maximum throughput achieved in 2D and 3D routers as a function of packet length. Routing the header phit takes one or two cycles more than forwarding a data phit due to the pass through the routing and arbitration stages. Thus, between each packet transfer the link will be idle for one or two cycles. Besides, short messages impose high demands into the arbitration process: the number of arbitrations increases as it is more likely that multiple header phits are simultaneously requesting multiple outputs. As expected, SAC shows the highest throughput followed by OAC.

Current networks deal with packets of less than 100 bytes and phit sizes in the order of 4 bytes. Thus, packets are in the range of 10 to 20 phits [18],[11]. We have selected a fixed packet length of 16 phits (64 bytes).

Table 2 shows the base latency (1-hop path) and maximum throughput for each alternative. OAC has the lower latency, due to its shorter node pipeline. As expected, the more flexible arbiter achieves the highest throughput and the more sequentialized arbiter, SIC, the lowest. However, the differences are not as significant as expected.

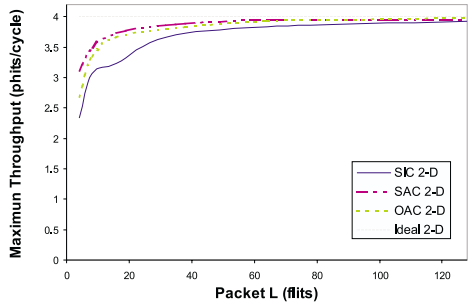

Figure 7: Maximum Throughput for 2-D node.

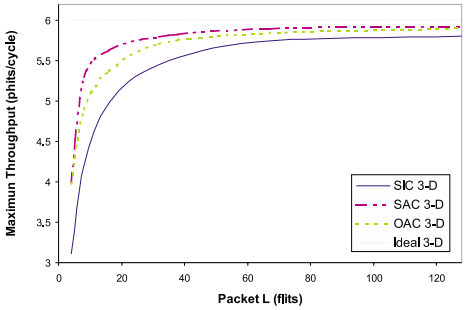

Figure 8: Maximum Throughput for 3-D node.

4 Network Analysis

This section is devoted to evaluate the impact of the routers in the network behavior under synthetic traffic. Synthetic loads are useful when the architect goal is to stress the network capabilities by considering that no other bottleneck exists on the entire machine except the subsystem under study. Besides, analysis of real traffic shows that many applications exhibit patterns that are quite close to synthetically generated patterns[24].

	OAC		**SAC**		**SIC**	
	2D	3D	2D	3D	2D	3D
1-hop Base Latency(cycles)	20	20	21	21	21	21
TH max (bytes/cycle)	14.5	22.0	15.0	22.7	13.0	20.0

Table 2: Single adaptive Bubble node performance for different arbiter designs.

We will later see, in Section 5, how the performance under synthetic traffic truly translates into system performance when executing real parallel applications.

The results presented below correspond to two torus networks with 64 nodes, an 8-ary 2-cube and a 4-ary 3-cube. We considered them under two different message destination patterns: random, and specific permutations. In the first case, all the nodes have the same probability of becoming the destination for a given message. However, two message length distributions were considered: short messages and bimodal which combines short and long messages. The latter resembles the traffic generated in DSM multiprocessors in which short messages correspond to requests (or invalidation primitives for cc-NUMA), and long messages correspond to cache lines.

For the specific permutation experiments, three traffic patterns were taken into account: matrix transpose, bit reversal, and perfect-shuffle. All three cases are frequently used in numerical applications, such as in ADI methods to solve differential equation systems and in FFT algorithms, among others. In all the experiments, the temporal traffic generation is uniform and randomly distributed over time.

Table 3 shows the maximum throughput of a 2D and 3D adaptive Bubble networks using each arbiter's implementation under different traffic patterns. The results in terms of phits/cycle are quite similar. SAC always outperfroms SIC and OAC due to its ability to match more than one request per cycle. The differences become more significant when considering the cycle (or arbitration) time. The OAC alternative, with the shortest arbitration time, clearly outperforms the other arbiters for any traffic pattern. Similar response is observed in a 3D network. In this case, SAC achieves better performance in phits per cycle for any of the three permutations when compared with OAC. Nevertheless, these minor benefits achieved by the most flexible arbiter, SAC, result in lower actual throughput due to their much higher implementation costs.

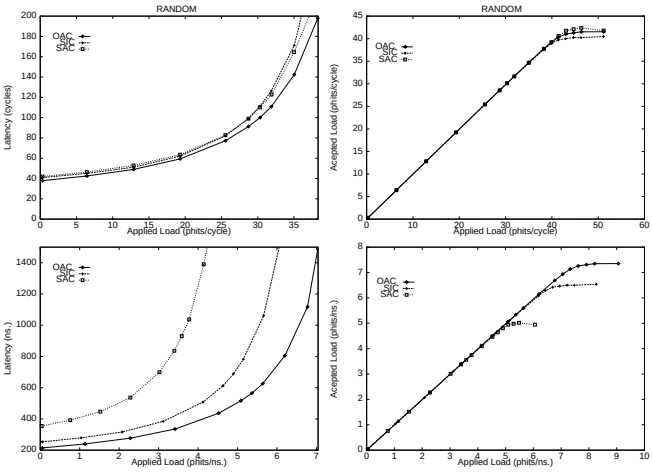

Figure 9: Latency and Throughput for a 64-node 2-D Torus under Random Traffic (a) Structural Performance and (b) Technological Performance.

These costs are also reflected in network latency. Firstly, because both SIC and SAC add one more stage to the node's pipeline. Secondly, the clock cycle differences results in large differences for packet latency under any traffic pattern. Figure 9 shows network performance from two different angles: from the architectural point of view and from the implementation's one. From the former point of view, there are minor gains achieved by OAC due to its better balanced pipeline. Notwithstanding, implementation constraints are not to be overlooked as they drive the final design.

Similar behavior was observed for all other traffic patterns in 2D and 3D topologies. For the sake of simplicity we will only show latency results, in Figures 10 and 11, for bimodal traffic and matrix transpose permutation.

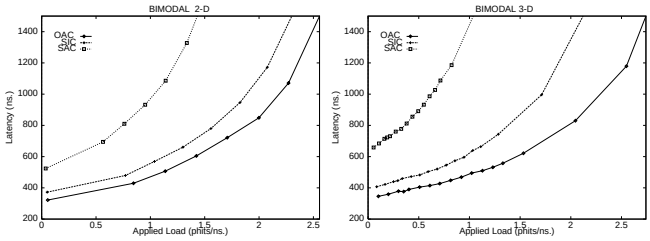

Figure 10: Latency and Throughput for a 64-node 2-D and 3-D Torus under Bimodal Traffic ($L_{big} = 10L_{small}$ with 0.1 probability).

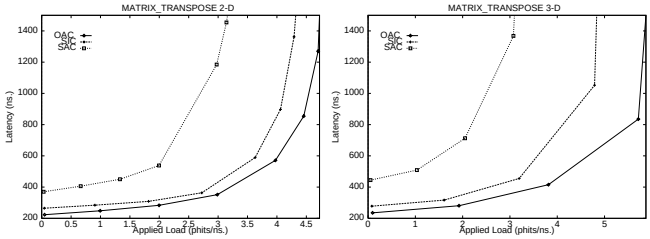

Figure 11: Latency and Throughput for a 64-node 2-D and 3-D Torus under Matrix Transpose Traffic.

5 System Performance Under Real Loads

Finally, we have evaluated the impact of the arbiter's design when executing real parallel applications. This evaluation is carried out by linking our RTL network simulator SICOSYS to Rsim [14], a DSM multiprocessor simulator. Figure 12 shows the main elements of this simulation environment. Rsim emulates a cc-NUMA architecture which is highly configurable. We have used the default values included in the Rsim distribution [15].

Three network configurations, one for each arbiter's alternative were used in these experiments. As previously mentioned, we set the channel width to 4 bytes. Assuming a cache line of 64 bytes and cache management commands of 16 bytes, this results in data packets of 20 phits and request and invalidation packets of 4 phits. Another important parameter is the speed of the processor relative to the network speed. We have

Traffic		Random		Bimodal		M. Transpose		P.Shuffle		Bit Reversal	
Arbiter		Phits/cycle	Phits/ns	Phits/cycle	Phits/ns	Phits/cycle	Phits/ns	Phits/cycle	Phits/ns	Phits/cycle	Phits/ns
2D	OAC	41.50	7.34	35.51	6.28	28.44	5.03	37.70	6.67	34.87	6.17
	SIC	40.47	6.54	31.50	5.08	28.00	4.55	35.24	5.69	32.57	5.26
	SAC	42.40	5.02	33.85	4.01	29.95	3.54	38.22	4.52	35.57	4.21
3D	OAC	50.30	8.00	43.08	6.86	44.51	7.19	45.40	7.29	41.58	6.66
	SIC	47.07	6.28	39.85	5.31	42.20	5.62	45.75	6.10	41.82	5.57
	SAC	50.56	4.33	42.48	3.63	48.12	4.12	49.04	4.19	44.41	3.80

Table 3: Maximum achievable throughput for 2-D and 3-D networks for the three arbiter designs under various traffic patterns.

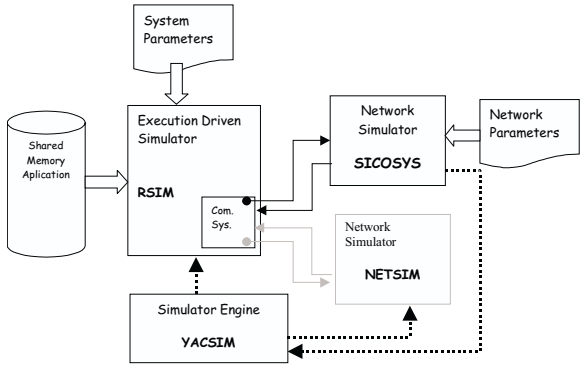

Figure 12: Execution Driven Simulator.

consider a 600 MHz processor, like the new release of the alpha microprocessor [12], which is the family used in the Cray T3E system [18]. Table 4 shows the relative speed of the processor with respect to the three networks, using the clock period previously calculated. Although these values highly depend on the technology used (0.7 μm), they are quite realistic. In fact, the Cray T3E system [18] has a relative processor speed approximately 8 times faster than the network.

Arbiter	Clock Speed(MHz) 2D	3D	Relative speed 2D	3D
OAC	177	159	3.39	3.78
SAC	162	133	3.70	4.51
SIC	118	85	5.08	7.05

Table 4: Relative speed of a 600 MHz processor node versus network speed.

Finally, we fed the simulator with two applications selected from the SPLASH [16] and SPLASH-2 [25] suites: Radix and MP3D, which had already been ported into Rsim by researchers at Rice University [17]. These two applications were selected because they have low data locality, so both are very demanding.

5.1 Performance Using Radix Benchmark

Radix is an integer sorting kernel described in [2]. Radix has three distinctive phases: data distribution, kernel computation and results verification. The permutation of keys generates an all-to-all communication pattern. In order to limit simulation time, we selected a problem size below the suggested by default. The integer set has 512 K elements and the radix is 1024. We also simplify key generation by providing the keys as an input file. This changes won't affect the metrics under consideration.

The kernel computation phase is the longest one, having execution times two order of magnitude larger than the other two phases altogether. Thus, we will present figures for this phase only.

Figure 13 shows the execution time for two 2D network sizes: 16 and 64 nodes. As expected, execution time decreases when increasing the number of nodes. This is because the communication to computation ratio is $P \div (P-1)$, being P the number of processors.

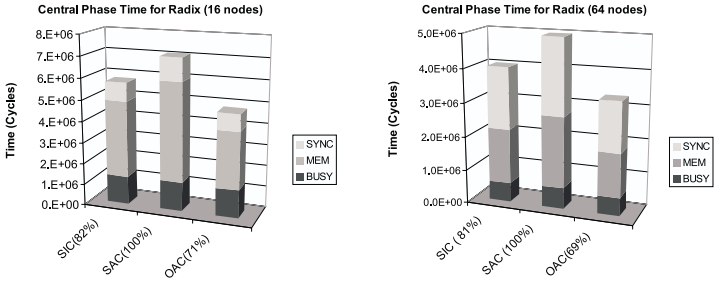

Figure 13: Radix execution time for (a) 4×4 Torus and (b) 8×8 Torus.

Obviously, the system with OAC network executes faster than with SIC or SAC networks. The SAC system, for 4×4 Torus (8×8 Torus), has an execution time 40% (48%) slower than the OAC system and the SIC system is 16% (20%) slower.

Relating these gains to the clock cycles and network performance estimated under synthetic loads is not a simple task from a quantitative point of view. Nevertheless, the trends exhibited under synthetic loads are qualitatively translated into the network behavior under real traffic patterns. Real loads differ from synthetic ones in that load patterns varies during execution, both in injection rates as well as on the selection of destinations. At low loads, the application is more sensitive to base latency. At high loads the application is more sensitive to network throughput. Thus, we need to observe the evolution of network traffic over time.

Figure 14 shows how network throughput (the number of data delivered per time unit) oscillates over time for OAC and SIC networks. We can identify two phases

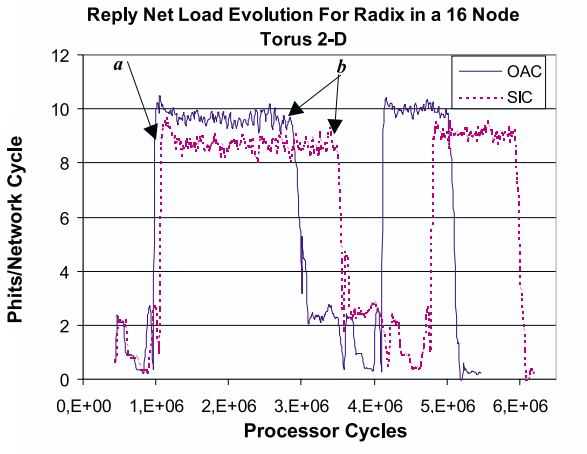

Figure 14: Reply net load evolution for Radix in a 4×4 torus.

which are throughput-bounded, with three other minor phases in between with low loads. The OAC network delivers 10 phits/cycle at saturation loads while SIC is only able to reach 8.5 phits/cycle. This reduces the throughput-bounded phase length significantly as seen in Figure 14 (end of phase has a *b* tag). Besides, the OAC system reaches this phase earlier than SIC due to its smaller network latency. The combined effect of lower latency and higher throughput results in a reduction of the execution time greater than that due to the relative speeds of both networks (SIC's clock is only 9% slower than OAC).

Finally, we have run Radix in a 3-D network ($4 \times 4 \times 4$) with the execution times shown in Figure 15. The impact of each router type is similar than for 2D. If we compare the two 64-node networks (Figure 13.b versus Figure 15) we can see that 2D networks reduce Radix's execution time for SAC and SIC when compared with their 3D counterparts.

On the other hand, the 3D OAC system presents the shortest execution time, slightly lower than for 2D. If we look back to Table 3 we see that both OAC and SIC exhibit higher throughput for 3D configurations.

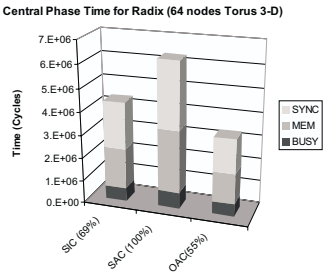

Figure 15: Radix execution time for $4 \times 4 \times 4$ Torus.

5.2 Performance Using MP3D Benchmark

MP3D [16] is a particle-based wind tunnel simulation used for aeronautical tests. The MP3D code presents no data locality, thus being quite sensitive to network delay. We selected the default problem size: 50,000 particles over a geometry of 2353 cells (test.geom[16]) but the number of iterations was reduced to 2 in order to limit our simulation time.

Similarly to Radix, we have simplified the initialization process which is not dependent on network response, and focus on the execution time of the 2 iterations. Figure 16 shows the execution time for the 16 node network size.

As expected, for 4×4 Torus the SAC system, has an execution time 49% slower than the OAC system and the SIC system is 17% slower as well.

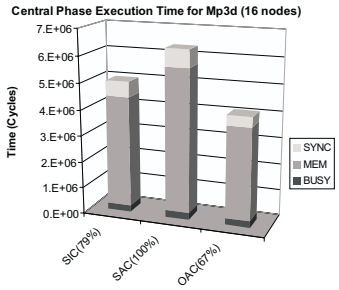

Figure 16: Mp3d execution time for 4×4 Torus.

Figure 17 shows how network throughput (the number of data delivered per time unit) oscillates over time for OAC and SIC. The graph presents a clear load pattern for each of the 2 iterations. Two phases are communication intensive: a wide phase with a delivered load around 5-7 phits/cycle and a narrow which reaches 8 phits/cycle in both networks. By comparing OAC and SIC we can see that they deliver similar throughput which indicates none of them reached loads above their saturation points. In this case, network performance at very low loads is critical. We can see that at 63 million cycles (initialization phase) both systems reach the end of the communication phase with little advantage for OAC. However, the application reaches the next communication intensive phase much earlier for OAC than for SIC. This indicates that the code executed in between was very sensitive to base latency which is lower for OAC as shown in sections 3 and 4.

6 Conclusions

The architectural design of a router cannot be carried out in isolation from either VLSI constraints or real applications demands. The implementation of an adaptive router motivated the low-level analysis of alternative adaptive arbiter's implementations.

In order to evaluate the impact of the arbiter's design at all levels in a parallel system a thorough methodology has been used. Firstly, we produced the detailed VLSI designs for an Output Arbiter Crossbar (OAC),

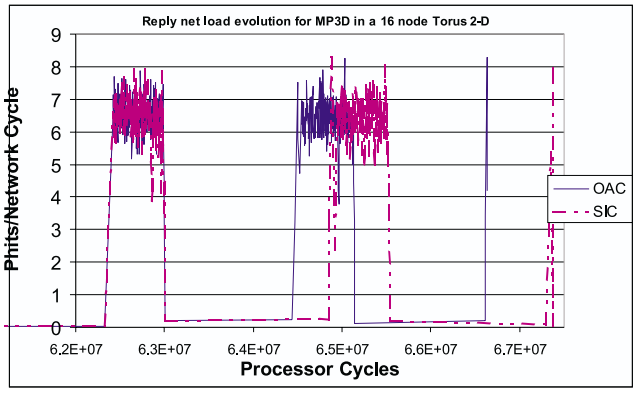

Figure 17: Reply net load evolution for MP3D in a 4×4 torus.

a Symmetric Arbiter Crossbar (SAC) and a Sequential Arbiter Crossbar (SIC). This approach provided a fair VLSI cost comparison of each arbiter under similar router conditions. Secondly, detailed simulation in RTL environments, was carried out to study the impact of arbitration in the node and network response. Finally, we evaluated the impact of this router element on execution time of two applications from the SPLASH-1/2 suites.

In single node experiments, the OAC arbiter exhibits the best performance in bytes/ns. Although SAC is able to improve the number of bytes/cycle, the OAC presents the best trade-off between number of input-to-output assignments per cycle and the arbitration time. Besides, the OAC router present a well balanced pipeline which has one stage less than the other two alternatives. Similarly, network performance is best for OAC for any synthetic pattern due to both its shorter pipeline and its higher clock frequency.

Parallel applications clearly benefit from the higher throughput and lower base latency exhibited by OAC. It is not easy to translate network performance under synthetic loads into direct gains in real system performance for a given application. But, by analyzing in more depth the communication demands of the two selected applications, the connections between synthetic and real loads can be drawn. Radix execution is significantly shorter for OAC due to its high network throughput. MP3D execution is affected by base latency instead. In any case, the impact of design decisions at low-level on the execution time of real applications is higher than it could be expected.

References

[1] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, October 1991.

[2] G. Blelloch et al.,"A Comparison of Sorting Algorithms for the Connection Machine CM-2", *In Proc. Symposium on Parallel Algorithms and Architectures*, pp. 3-16, July de 1991.

[3] [3] J. Carbonaro and F. Verhoorn, "Cavallino: The teraflops router and NIC," *Hot Interconnects Symposium IV*, August 1996.

[4] C. Carrión et al., "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," *Int. Conference on High Performance Computing*, pp. 322-329, India, December,1997.

[5] A.A. Chien, "A cost and speed model for k-ary n-cube wormhole routers," *Interconnects'93*, Palo Alto, California, August 1993.

[6] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Computers*, vol. C-36, no. 5, pp. 547-553, May 1987.

[7] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. on Computers*, vol. C–39, no. 6, pp. 775–785, June 1990.

[8] [8] W.J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, March 1992.

[9] W. J. Dally, et al., "Architecture and implementation of the Reliable Router," *Hot Interconnects Symposium II*, August 1994.

[10] J. Duato,"A new theory of deadlock-free adaptive routing in wormhole networks." *IEEE Trans. on Parallel and Distributed Systems*, vol.4, no.12, pp.1320-1331, December 1993.

[11] M. Galles, "Scalable pipelined interconnect for distributed endpoint routing: The SPIDER chip," *Hot Interconnects Symposium IV*, pp. 141-146, August 1996.

[12] R. E. Kessler et al., "The Alpha 21264 Microprocessor Architecture", in *Proc. of Int. Conference of Computer Design*, October 1998.

[13] J. Laudon and D. Lenoski, "The SGI origin: A cc-NUMA highly scalable server," *Int. Symposium on Computer Arch.*, pp. 241-251, June 1997.

[14] V. S. Pai et al. "Rsim: An execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors", *IEEE TCCA Newsletter*, October 1997.

[15] V. S. Pai et al., "Rsim Reference Manual. Version 1.0". Department of Electrical and Computer Engineering, Rice University. Technical Report 9705. July 1997.

[16] [16] J. Pal Singh et al. "SPLASH: Stanford Parallel Applications for Shared-Memory". In Computer Architecture News, vol. 20, no. 1, pp 5-44, May 1992.

[17] SPLASH/SPLASH2 applications/kernels ported for Rsim available at http://www.ece.rice.edu/rsim/ Distributions/splash_apps.tar.gz.

[18] S. L. Scott and G. Thorson, "The Cray T3E network: Adaptive routing in a high performance 3-D torus", *Hot Interconnects Symp. IV*, pp. 147-155, August 1996.

[19] J.M. Prellezo et al., "SICOSYS: a interconnection network simulator for parallel computers," Technical Report TR-ATC2-UC98, June 1998.

[20] V. Puente et al., "Adaptive Bubble Router: a Design to Balance Latency and Throughput in Networks for Parallel Computers", Technical Report TR-ATC3-UC98, September 1998.

[21] Y. Tamir and C. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches", *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13-27, January 1993 .

[22] C.L. Seitz and W.K. Su, "A Family of Routing and Communication Chips Based on the Mosaic" *Proc. of the 13th Symp. on Integrated Systems*, The MIT Press, pp. 332-337, January 1993.

[23] *Synopsys On Line Documentation* Version 1997.08, 1997.

[24] A.S. Vaidya, A. Sivasubramaniam and C.R. Das, "Performance benefits of virtual channels and adaptive routing: An application-driven study," *Int. Conf. on Supercomputing*, pp. 140-147, July, 1997

[25] S. C. Woo et al,."The SPLASH-2 Programs: Characterization and Methodological Considerations". In Proc. of the *22nd Int. Symposium on Computer Arch.*, pp. 24-36. June 1995.