Optimal Topology for Distributed Shared-Memory Multiprocessors: Hypercubes Again?

José Duato and M.P. Malumbres

Facultad de Informática, Universidad Politécnica de Valencia
P.O.B. 22012, 46071 - Valencia, SPAIN. E-mail: {jduato,mperez}@gap.upv.es

Abstract. Many distributed shared-memory multiprocessors (DSM) use a direct interconnection network to implement a cache coherence protocol. An interesting characteristic of the message traffic produced by coherence protocols is that all the messages are very short. Most current multicomputers use low dimensional meshes or tori because these topologies usually achieve a higher performance. However, when messages are very short, latency is mainly dominated by the distance traveled in the network. As a consequence, higher dimensional topologies may achieve a lower latency than low-dimensional topologies. In this paper, we compare the 2D-mesh and the hypercube topologies assuming a very detailed router model. Network load has been modeled taking into account the traffic produced by cache coherence protocols. Performance results show that average latency for hypercubes is slightly lower than for meshes. Moreover, hypercubes achieve a much higher throughput than meshes, making them suitable for DSMs.

1 Introduction

Distributed shared-memory multiprocessors (DSM) are gaining acceptance because they are easier to program than multicomputers. Recently proposed DSMs use a direct interconnection network to access remote memory locations, making these architectures scalable [12, 11]. Most DSM implement a cache coherence protocol. An interesting characteristic of the message traffic produced by coherence protocols is that all the messages are very short. In particular, invalidations and acknowledgments require a single flit. Messages containing a cache line usually range from four to sixteen flits depending on flit width and line size.

Most current multicomputers like Intel Paragon and Cray T3D use wormhole switching [4] and low dimensional meshes or tori [10, 9]. Low-dimensional topologies allow the use of wider channels, therefore increasing channel bandwidth [1]. This is important when messages are long. However, when messages are very short, latency is mainly dominated by the distance traveled in the network. As a consequence, higher dimensional topologies may achieve a lower latency than low-dimensional topologies. Taking into account that invalidation protocols require a very low message latency to work efficiently, the optimal topology for DSMs may differ from the one for multicomputers. Throughput is also important for cache

This work was supported by the Spanish CICYT under Grant TIC94–0510–C02–01

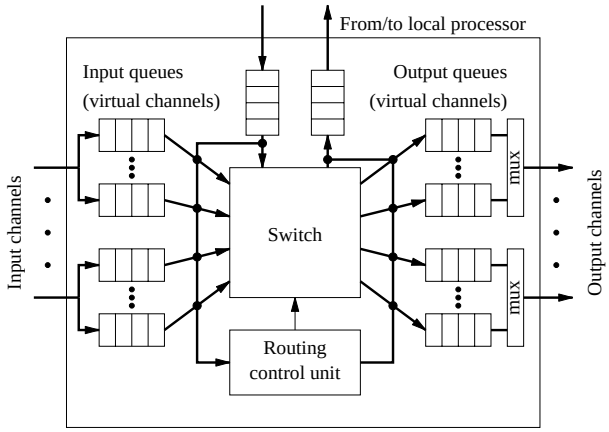

Fig. 1. Router model used in our study

coherence protocols, especially for update protocols. Again, high-dimensional topologies may achieve a higher throughput for short messages because they offer more alternative paths, especially when adaptive routing is used.

In this paper, we present a performance evaluation of deterministic and fully adaptive routing algorithms for meshes and hypercubes. Performance evaluation uses a very detailed router model. The design complexity of the router has been considered while computing internal delays. Also, wire length has been taken into account to compute channel delay and bandwidth. Network load has been modeled taking into account the traffic produced by cache coherence protocols in DSMs. In section 2, we describe the router model that we have considered. In section 3, we present the network traffic pattern. In section 4 we present the simulation results for different approaches. Finally, conclusions are drawn.

2 Router Model

Each router consists of a routing control unit, a switch, and several channels. Figure 1 shows the router model used in our simulator. The routing control unit selects the output channel for a message as a function of its destination node, the current node and the output channel status. The routing control unit can only process one message header at a time. We also considered an alternative router design in which each virtual channel has an independent routing control unit, allowing a concurrent processing of message headers that arrive at the node. The switch is a crossbar. So, it allows multiple messages traversing it simultaneously without interference.

Physical channels can be split into several virtual channels. Virtual channels are assigned to the physical link using a demand-slotted round-robin arbitration scheme. Each virtual channel has an associated buffer. This buffer is divided into two halves, one associated with the output port of the switch (m flits), and

another one associated with the input to the next node's switch (n flits). We will use the $m+n$ notation to specify this buffer size. We have used a variable buffer size to study the impact on performance.

We assume that all operations inside each router are synchronized by its local clock signal. To compute the clock frequency of each router, we use the delay model proposed in [3]. It assumes 0.8 micron CMOS gate array technology for the implementation. The delay of each component is computed as follows:

- Routing control unit. Routing a message involves the following operations: Address decoding, routing decision, and header selection. According to [3], the address decoder delay is equal to 2.7 ns. The routing decision logic has a delay that grows logarithmically with the number of alternatives, or degree of freedom, offered by the routing algorithm. Denoting by F the degree of freedom, this circuit has a delay given by $0.6 + 0.6 \log F$ ns. Finally, the routing control unit must compute the new header, depending on the output channel selected. This operation has a delay given by $1.4 + 0.6 \log F$ ns. Therefore, total routing time will be the sum of all the delays, yielding:

  $T_r = 2.7 + 0.6 + 0.6 \log F + 1.4 + 0.6 \log F = 4.7 + 1.2 \log F$ ns.

- Switch. The time required to transfer a flit from one input channel to the corresponding output channel is the sum of the delay involved in the internal flow control unit, the delay of the crossbar, and the set-up time of the output channel latch. The flow control unit has a constant delay equal to 2.2 ns. The crossbar delay grows logarithmically with the number of ports. Assuming that P is the number of ports of the crossbar, its delay is given by $0.4 + 0.6 \log P$ ns. Finally, the set-up time of a latch is 0.8 ns. Therefore, switch time is:

  $T_s = 2.2 + 0.4 + 0.6 \log P + 0.8 = 3.4 + 0.6 \log P$ ns.

- Channels. The time required to transfer a flit across a physical channel includes the off-chip delay across the wires, and the time required to latch it onto the destination. The latter time is the sum of input buffer, input latch and synchronizer delays. Typical values for the technology used are 0.6, 0.8, and 1.0 ns, respectively, yielding 2.4 ns per flit. The off-chip delay across the wires depends on their length. In particular, topologies like 2D-meshes have constant wire length. For the technology used, assuming 25 pF load, typical propagation delay across wires is 1.5 ns. However, hypercubes have wires with different lengths. So, wire delay must be computed for hypercubes taking into account wire length in order to make a fair comparison with mesh topologies. For example, if we have an 8D-hypercube, we can assemble the topology in three dimensions as shown in Figure 2. As can be seen, the shortest wires have the same length that they would have in a 2D-mesh. Also, there are some wires twice as long as the shortest ones. Finally, the longest wire is four times the size of the shortest one. Thus, the off-chip delay in an 8D-hypercube will depend on wire length. For the shortest wires, a typical value will be 1.5 ns (the same as for 2D-meshes). For the remaining wire lengths, the off-chip delay will be 3 ns and 6 ns, respectively.

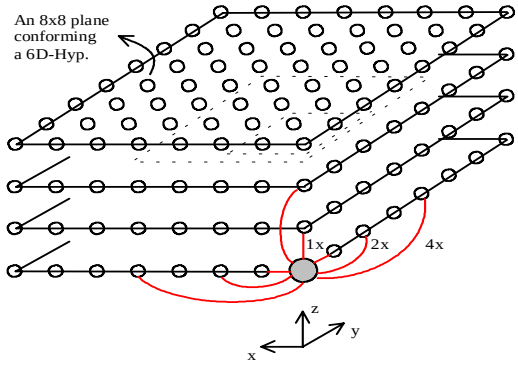

Fig. 2. A three-dimensional implementation of a 8D-hypercube

If virtual channels are used, the time required to arbitrate and select one of the ready flits must be added. The virtual channel controller has a delay logarithmic in the number of virtual channels per physical channel. If V is the number of virtual channels per physical channel, virtual channel controller delay is $1.24 + 0.6 \log V$ ns. If we denote the off-chip delay for the shortest wires as T_{c1}, and the off-chip delays for wires that are twice and four times longer as T_{c2} and T_{c4}, respectively, total channel delay yields:

$T_{c1} = 2.4 + 1.5 + 1.24 + 0.6 \log V = 5.14 + 0.6 \log V$ ns.
$T_{c2} = 2.4 + 1.5 * 2 + 1.24 + 0.6 \log V = 6.64 + 0.6 \log V$ ns.
$T_{c4} = 2.4 + 1.5 * 4 + 1.24 + 0.6 \log V = 9.64 + 0.6 \log V$ ns.

In what follows, we compute the value of F and P as a function of the topology and the number of virtual channels for deterministic and adaptive routing algorithms. The deterministic routing algorithm is the dimension-order routing algorithm. The adaptive routing algorithm is based on Duato's theory [7]. In this routing algorithm, all the virtual channels but one are used for fully adaptive minimal routing. The remaining virtual channel is used to avoid deadlocks by routing in dimension-order.

1. 2D-mesh with deterministic routing. A deterministic routing algorithm has a single routing choice. However, messages can use any virtual channel. If there are V virtual channels per physical channel then $F = V$. Also, each crossbar has $P = 4V + 1$ ports, including the port connecting to the local processor.

2. 2D-mesh with adaptive routing. The number of routing choices is equal to $F = 2(V - 1) + 1$, because we have $V - 1$ virtual channels in each dimension that can be used to cross the dimensions in any order plus one additional virtual channel to avoid deadlock [7]. As above, the number of ports is given by $P = 4V + 1$.

3. 8D-hypercube with deterministic routing. In this case, we have $F = V$ and $P = 8V + 1$.

4. 8D-hypercube with adaptive routing. The number of routing choices is $F = 8(V - 1) + 1$ and the number of ports is $P = 8V + 1$.

3 Network Traffic Pattern Description

We assume a distributed shared-memory multiprocessor architecture. So, traffic consists of two kinds of messages: control messages containing invalidations and acknowledgments (1 data flit), and data messages that transport cache lines (8 data flits). Control messages represent a 60% of the total number of messages and may have more than one destination (multicast messages). The rest of messages will be unicast data messages, following the usual distribution in DSMs.

For each simulation run, we have considered that message generation rate is constant and the same for all the nodes. Once the network has reached a steady state, the flit generation rate is equal to the flit reception rate (traffic). We have evaluated the full range of traffic, from low load to saturation. The message destination is randomly chosen among all the nodes. This pattern has been widely used in other performance evaluation studies [6, 2].

4 Simulation Results

We evaluated 2D-mesh and hypercube topologies with 256 nodes, considering different routing algorithms and network parameters, and assuming the specific traffic pattern described above. We ran simulations using the minimum number of virtual channels per physical channel and the minimum plus one. The evaluation methodology used is based on the one proposed in [7]. The most important performance measure in DSMs is latency (time required to deliver a message). So, plots will represent latency versus network traffic. Traffic is the flit reception rate. Latency is measured in nanoseconds. Traffic is measured in flits per node per microsecond. Note that we have used absolute measurement units because we are going to compare routing algorithms that involve different implementations, and consequently, different clock frequencies. When simulation results do not consider the router implementation, the latency and traffic will be measured in clock cycles and flits per node per cycle, respectively.

4.1 Effect of queue size and multiple routing control units

We analyzed the effect of flit queues with capacity for $2 + 2$ and $4 + 5$ flits. Note that a queue with capacity for $4 + 5$ flits is able to store a whole message, including its header. Also we considered two router models: one of them has the typical routing control unit in which header flits are routed sequentially, and the other one is able to make routing operations in parallel. So, we can determine if the routing control unit is a bottleneck when short messages are used.

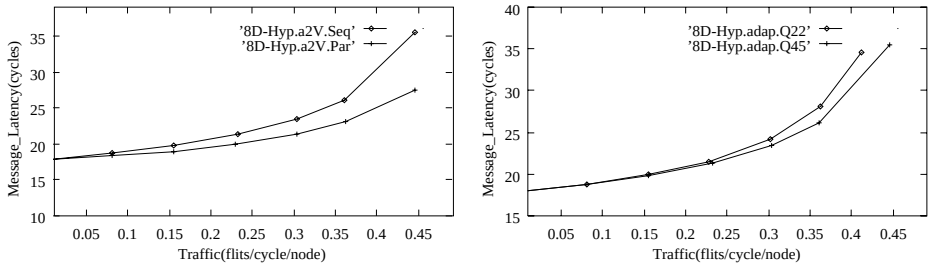

Fig. 3. Single vs multiple routing control units, and effect of queue size in 8D-hypercubes using an adaptive routing algorithm

Figure 3a shows the difference between using single and multiple routing control units in an 8D-hypercube with adaptive routing. As can be seen, latency is almost identical when traffic load is low. When traffic increases, the difference between both approaches also increases, achieving an improvement of up to 25% in average latency. In general, the latency improvement is not worth the additional cost of replicating the routing control units. Similar results are obtained for 2D-mesh topologies. So, in what follows, a single routing control unit will be used. Figure 3b shows the effect of using flit queues of different capacities. This effect is very small. So, in what follows, we will use a 4+5 queue size per channel.

4.2 Considering router delays and implementation constraints

The most important implementation constraints proposed in the literature are pin count and network bisection width [1]. For small and medium size networks, channel width is only limited by pin count, existing proposals to increase the utilization of the available bisection bandwidth without increasing pin count [5].

In order to make a fair comparison between the 2D-mesh and the hypercube, both topologies should have the same pin count. So, physical channels are twice as wide in the 2D-mesh than in the 8D-hypercube. Thus, the 8D-hypercube will spend 2 clock cycles to transfer one flit across a short physical channel, while the 2D-mesh will only require one cycle. Also, depending on parameters like number of virtual channels, routing algorithm, network dimensions and physical layout, we can compute the delays for all the router components. Using the expressions obtained in the router model section, we present in table 1 the delays and clock periods for 2D-mesh and 8D-hypercube topologies.

2D-mesh	T_r	T_s	T_{c1}	T_{clk}
Det-1V	4.7	4.79	5.14	5.14
Det-2V	5.9	5.3	5.74	5.9
Adap-2V	6.6	5.3	5.74	6.6
Adap-3V	7.49	5.62	6.09	7.49

8D-hyp	T_r	T_s	T_{c1}	T_{c2}	T_{c4}	T_{clk}
Det-1V	4.7	5.3	5.14	6.64	9.64	5.3
Det-2V	5.9	5.85	5.74	7.24	10.24	5.9
Adap-2V	8.5	5.85	5.74	7.24	10.24	5.85
Adap-3V	9.6	6.2	6.09	7.59	10.59	6.2

Table 1. Delays for routing control unit (T_r), switch (T_s) and channels of different lengths (T_{ci}), and router clock period (T_{clk}) in 8D-hypercube and 2D-mesh topologies.

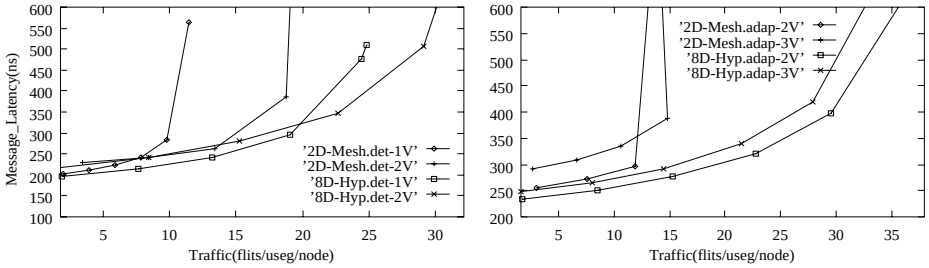

Fig. 4. Performance of deterministic and adaptive routing algorithms for 2D-mesh and 8D-hypercube when constant pin count and variable channel length are considered

In the 2D-mesh, the delays for the routing control unit, switch and channels are very similar. So, the clock period is determined by the slowest component.

However, in 8D-hypercubes the delay for long channels can be up to twice the switch delay. Delays for the switch and routing control unit are very similar when deterministic routing is used. However, these delays considerably differ for adaptive routing. So, we have chosen clock period to be equal to $\max(T_r, T_s)$ for deterministic routing, and equal to T_s for adaptive routing. As a consequence, the routing operation will require two clock cycles when adaptive routing is used. Taking into account that transferring one flit across a short physical channel requires 2 clock cycles (due to pin count constraints), transferring one flit across the rest of the channels can be done in four clock cycles.

Figure 4a shows simulation results for deterministic routing algorithms with one and two virtual channels, assuming the timing considerations mentioned above. Both topologies achieve similar average message latency when network load is low, being slightly better for the hypercube with one virtual channel. However, the hypercube increases throughput with respect to the 2D-mesh by a factor of 2 and 1.5 for one and two virtual channels, respectively.

Figure 4b shows simulation results for adaptive routing algorithms. As can be seen, the hypercube offers better performance than 2D-meshes, achieving a 25% reduction in message latency for three virtual channels, and a 10% reduction for two virtual channels. Moreover, in the latter case, the hypercube improves throughput by a factor of 2.7 over the 2D-mesh. On the other hand, performance does not improve when the number of virtual channels is increased from two to three. This result was already presented in [8].

5 Conclusions

Previous research has shown that low-dimensional topologies achieve higher performance than high-dimensional topologies. While this is true for multicomputers, previous research efforts did not consider the particular characteristics of network traffic in DSMs. When cache coherence protocols are implemented in hardware, messages are very short. In this case, latency heavily depends on the distance between nodes, therefore favoring the use of high-dimensional topologies.

In this paper, we have evaluated 2D-meshes and 8D-hypercubes using traffic patterns that are typical in DSMs. We have used deterministic and fully adaptive routing algorithms. Also, several network parameters, like number of virtual channels, buffer sizes and router models have been evaluated. To make a fair comparison, we have taken into account the delays of the main components of the router, as well as pin count constraints and variable wire lengths. The simulation results show that the 8D-hypercube behaves better than the 2D-mesh in all the cases, achieving a slightly lower latency and a much higher throughput. Also, the maximum performance is obtained for both topologies when the minimum number of virtual channels is used.

In summary, hypercubes perform better than 2D-meshes when traffic consists of very short messages, and router designs are optimized for each topology. Note however that we did not analyze very large networks. Bisection width may be the main constraint for channel width in networks with thousands of processors. So, results may change for large networks. Anyway, current DSMs have a few hundreds of nodes at most.

References

1. A. Agarwal, "Limits on interconnection network performance", *IEEE Trans. Parallel Distributed Syst.*, vol. 2, no. 4, pp. 398–412, Oct. 1991.
2. R.V. Boppana, and S. Chalasani, "A comparison of adaptive wormhole routing algorithms," in *Proc. 20th Annu. Int. Symp. Comput. Architecture*, May 1993.
3. A.A. Chien, "A cost and speed model for k-ary n-cube wormhole routers," in *Proc. Hot Interconnects'93*, Aug. 1993.
4. W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, May 1987.
5. W.J. Dally, "Express cubes: Improving the performance of k-ary n-cube interconnection networks," *IEEE Trans. Comput.*, vol. 40, no. 9, pp. 1016–1023, Sept. 1991.
6. W.J. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel Distributed Syst.*, vol. 3, no. 2, pp. 194–205, Mar. 1992.
7. J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 4, no. 12, pp. 1320–1331, Dec. 1993.
8. J. Duato and P. López, "Performance evaluation of adaptive routing algorithms for k-ary n-cubes," in *Proc. Workshop on Parallel Computer Routing and Communication*, May 1994.
9. Intel Scalable Systems Division, "*Intel Paragon Systems Manual*," Intel Corporation.
10. R.E. Kessler and J.L. Schwarzmeier, "CRAY T3D: A new dimension for Cray Research," in *Compcon*, pp. 176–182, Spring 1993.
11. J. Kuskin et al., "The Stanford FLASH multiprocessor," in *Proc. 21st Annu. Int. Symp. Comput. Architecture*, April 1994.
12. D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz and M. Lam, "The Stanford DASH multiprocessor," *IEEE Computer Magazine*, vol. 25, no. 3, pp. 63–79, March 1992.

This article was processed using the LaTeX macro package with LLNCS style