Allocators and Review Session

Review Session: Thursday, 29 April 1999
Lecturer: Li-Shiuan Peh
Scribe: Li-Shiuan Peh (lspeh@cs.stanford.edu)

1 Topics covered in lecture and review session

Allocators

- Coupled-set of resources
- Separable Allocators
- Iterative Allocators
- Dally's Heuristic

Review Session

- Deadlock
- Flow Control
- Topology

2 Allocators

This section is based on Professor Dally's lecture notes.

There are two kinds of allocation, one is where each request requests for 1 out of a set of resources, such as in the case of the VC Allocator, where each input virtual chaannel wants one out of a set of output virtual channels. The other case is whereby each request wants a coupled set of resources, such as in the Switch Allocator, where each input virtual channel requests for an input port and an output port.

For the first kind of allocation, Prof. Dally showed in the previous lecture that it can be solved using a maximum bipartite matching algorithm on the requests graph. It was also shown that a matrix can be used to visualize the requests.

For the second kind of allocation whereby a coupled set of resources are requested, the same techniques can be used. Figure 1 shows an example of a switch allocator where there are 8 virtual channels, and 4 input and output ports on the switch. The requests can be drawn as a graph, as shown in Figure 2, and similarly solved using the maximum

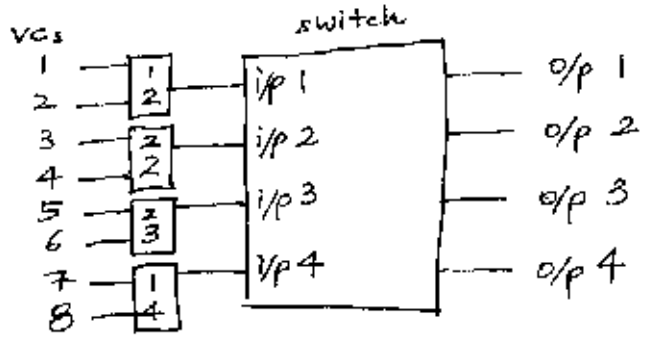

Figure 1: An example of a switch allocator.

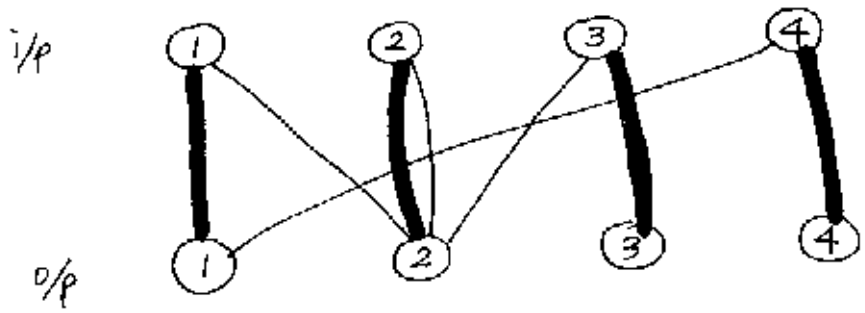

Figure 2: The switch allocation example represented as a graph. The bold lines are the allocations.

bipartite matching algorithm, since the goal is to obtain a single edge from each node. The only difference is that instead of input virtual channels and output virtual channels, the nodes correspond to input ports and output ports.

Allocators working with a coupled set of resources can similarly be viewed as a matrix, where the rows correspond to input ports, and columns correspond to output ports :-

	1	2	3	4
1	X	X		
2		X[1]		
3		X	X	
4	X			X

[1]Both virtual channels going into input port 2 are requesting for the same output port 2. Hence, a

This matrix visualization method can be extended to 3-dimensions, when there are 3 coupled resources. The matrix will then be a 3-D matrix, like a cube.

2.1 Separable Allocator

The separable allocator goes through each row, do a quick arbitration, and then goes through each column and do a quick arbitration. After arbitration of rows, the above example will be :-

	1	2	3	4
1	*X	X		
2		*X		
3		*X	X	
4	*X			X

And after arbitration of columns, the allocations will be :-

	1	2	3	4
1	*X	X		
2		*X		
3		X	X	
4	X			X

In this case, the allocation is rather bad since only two input ports were successful. Since the separable allocator does not rely on any global knowledge of what other input ports are requesting for, it is unable to base decisions on other information, and can only arbitrarily select one of the requests. It is a very fast allocator though.

2.2 Iterative Allocator

This algorithm was proposed by Anderson, Owicki, Saxe and Thacker. It is a variation of the Separable Allocator, whereby the separable allocation algorithm is iterated over several phases. So the row and column arbitrations are done for a single phase, and then, successful allocations are held, and failed requests are redone. Continuing the previous example, the failed requests from input ports 3 and 4 will be re-sent to output ports 3 and 4 (their original requests to output ports 1 and 2 are no longer considered since those ports have been allocated to input ports 1 and 2). A optimum allocation will occur in this case :-

single request can represent both, with either arbitrarily chosen should this request be successful.

	1	2	3	4
1	*X	X		
2		*X		
3		X	*X	
4	X			*X

The disadvantage of the iterative allocator is its speed, as it may iterate many times before all failed requests are re-tried. Besides, it's sub-optimal.

2.3 Dally's Heuristic Allocator

In practical routers, allocators must frequently be fast and allocations have to be done within a cycle. Hence, fast heuristics are usually used. Prof. Dally implemented this heuristic in routers he built in the past, and it worked pretty well in practice.

The heuristic is as follows :-

request Input ports send requests to the output ports

count Each output port counts the number of input requests to it and return the number to the input ports.

select Each input port selects the request which has the lowest number, i.e., which has the least number of other competing requests, and send that one request to the output port.

arbitrate Output ports then accept requests, and should there be more than one, do a simple arbitration.

Figure 3 shows Dally's heuristic applied to the example. In this case, the allocation is optimum :-

If we add an additional request from input port 2 to output port 4, there will be a need for the last arbitration phase at output port 4. A student raised the observation that this heuristic may result in everyone avoiding the most contended output port and hence end up idling that port. A counter-proposal was made by another student, who proposed iterating this heuristic twice, and thus solving the "busiest-port-turned-idle" problem.

This heuristic is rather fast, since it only requires 3 traversals of the switch fabric.

3 Review Session

3.1 Deadlock

Prof Dally's intuitive interpretation of Duato's algorithm was explained and presented, followed by 2 examples previously done in class and a new example obtained from the book.

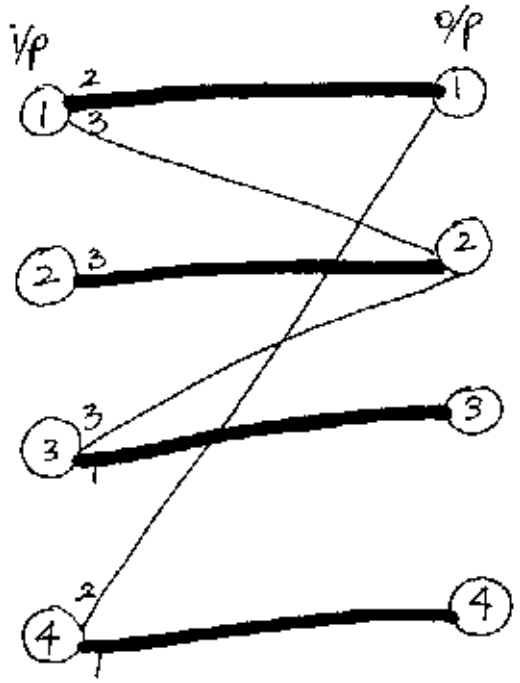

Figure 3: Dally's heuristic applied to the switch allocator example. The bold lines are the allocations.

3.1.1 Intuitive Interpreation of Duato's Algorithm

Given an adaptive R routing function which can be cyclic, and uses C channels of the network, if we can define a routing subfunction R1 which uses C1, a subset of the C channels, and R1 is fully connected and deadlock-free, and *R does not violate R1's ordering*, then, the network is deadlock-free.

Hence, given a network with R and R1, if we can find a path taken by a *single* packet such that this packet traverses a channel A of C1, and then traverses some channels of C2 (= C-C1), and back to a channel B of C1, then there arises a dependency from A to B. If $A- > B$ violates the ordering of the R1 subfunction, the network is not deadlock-free. Otherwise, it is deadlock-free.

3.1.2 Example 1: Dimension-Ordered Routing + Minimal Fully Adaptive Routing

In this example, R is a minimal fully-adaptive routing algorithm, and R1 is a dimension-ordered routing algorithm. The ordering for dimension-ordered routing is shown in Figure 4. Dimension-ordered routing dictates that once a packet uses a North or South

channel, it can't turn to any East or West channels. Hence, we can assign a number to each channel in the network, such that only traversals from lower-numbered channels to higher-numbered channels are allowed.

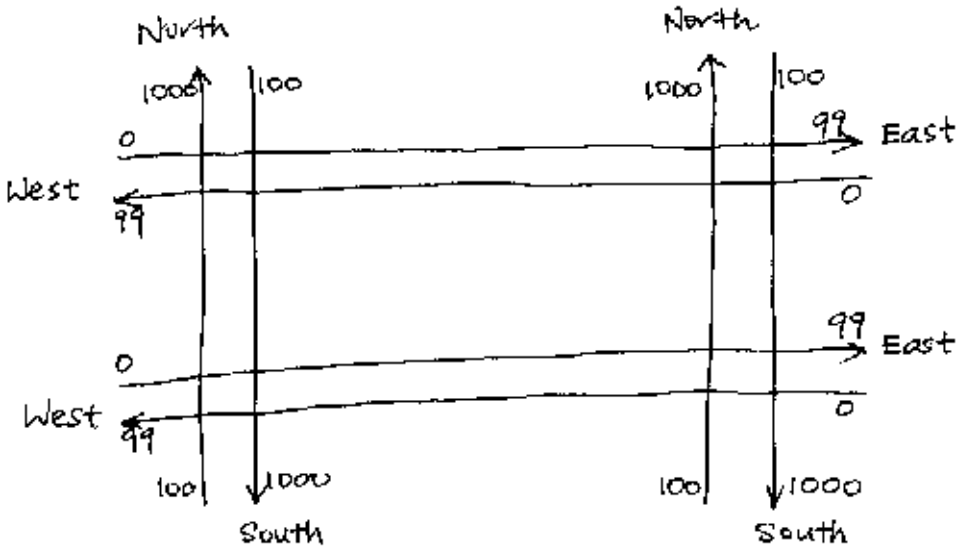

Figure 4: Dimension-ordered routing, with each channel in the network assigned a number, such that only traversals from lower-numbered channels to higher- numbered channels are allowed by the routing algorithm.

So, Figure 5 shows the path traversed by a single packet through channel A of C1, C2 channels, and then channel B of C2, thus creating a dependency from $A- > B$. Comparing this ordering to that of dimension-ordered routing in Figure 4, it can be seen that the ordering is not violated. Hence, the network is deadlock-free.

3.1.3 Example 2: Dimension-Ordered Routing + Non-minimal Fully Adaptive Routing

In this example, R is a *non*-minimal fully-adaptive routing algorithm, and R1 is still a dimension-ordered routing algorithm. Again, we start by finding a path traversed by a single packet through channel A of C1, then C2 channels, and then back to channel B of C1. This path is illustrated in Figure 6.

So, it's apparent that a dependency is created ($A- > B$) and this dependency violates the ordering that was inherent in dimension-ordered routing, since as shown in Figure 4, the eastward channels on the left must have a lower number than eastward channels to the right, so as to allow traversals in the X+ direction.

Hence, this network is not deadlock-free.

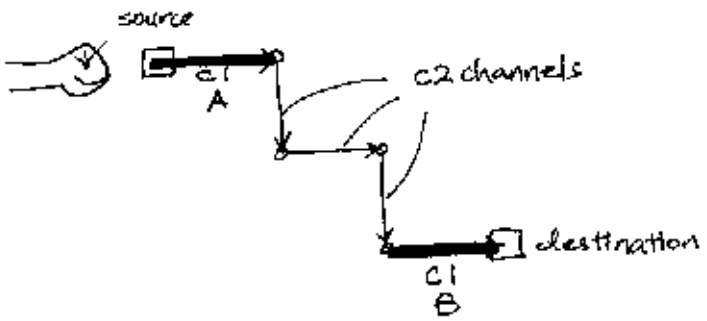

Figure 5: Example 1: R=minimal full-adaptive routing, R1=dimension-ordered

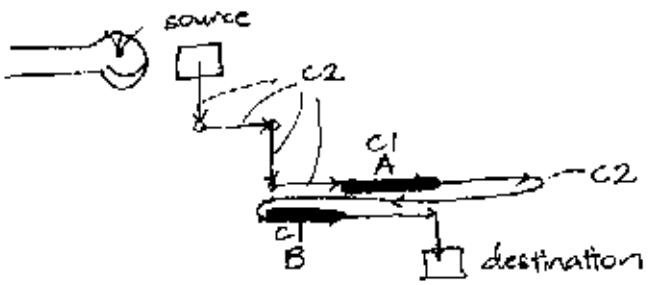

Figure 6: Example 2: R=non-minimal fully-adaptive routing, R1=dimension-ordered

3.1.4 Example 3: Minimal North-Last Routing + Minimal Fully Adaptive Routing

This example is from an exercise in the DYN text. In the text, it is solved with the formal approach of drawing all the different kinds of dependencies which are present in the channel dependency graph and seeing if there exists a cycle. During the review session, Prof. Dally's intuitive interpreation approach is used to analyze this network instead.

So R is a minimal fully-adaptive routing algorithm, whilst R1 is a minimal north-last routing algorithm. R1 is deadlock-free, since this is one of those deadlock-free routing algorithms derived from the turn model.

First, the ordering of the north-last routing algorithm is visualized in the numbering of channels in Figure 7. The north-last routing algorithm seems to only establish an ordering that the north direction must be used last, and hence, in our numbering method, that

the north channels have to have the largest numbers assigned. However, there is also an implicit ordering imposed. The difference between minimal north-last routing and dimension-ordered routing is that for dimension-ordered routing, once you get on the south channels, you can't disembark and move onto east or west channels. For minimal north-last routing, you can, so long as it's a minimal east or west channel. Thus, as is apparent in Figure 7, the rows above must have a lower number than rows below, since the routing algorithm allows traversals of east, followed by south, and then east channels.

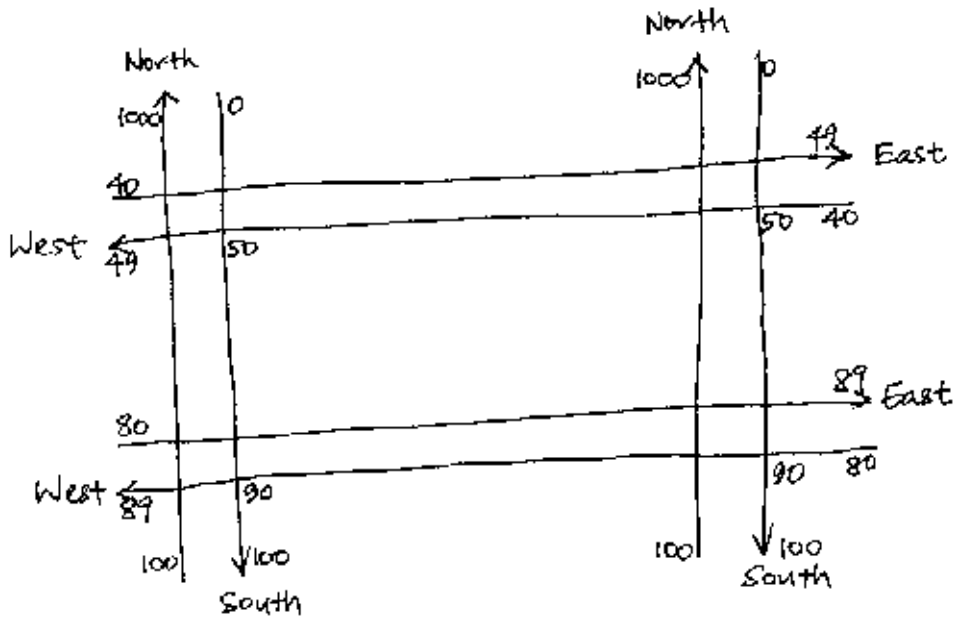

Figure 7: Ordering of minimal north-last routing

After establishing the ordering of the R1 subfunction, the next step consists of finding a path through which a single packet traverses channel A of C1, followed by C2 channels, and channel B of C1. This is shown in Figure 8. A dependency $A- > B$ is shown, and this violates the ordering of the north-last routing subfunction, since channel A, from a lower row of the network, should have a higher number than channel B from a higher row of the network. Thus $A- > B$ violates the lower to higher numbering, and the network is not deadlock-free.

3.2 Flow Control

The review session addressed the trade-offs arising from each flow-control technique, as compared to its predecessor. Store-and-forward (SAF), Virtual Cut-through (VCT), Wormhole (WH), Virtual-Channel (VC) and Flit-Reservation (FR) are reviewed, looking

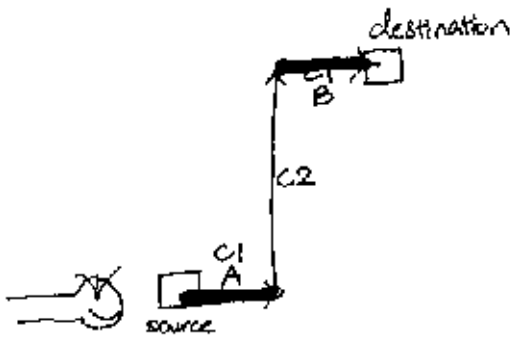

Figure 8: Example 3: R=minimal fully-adaptive routing, R1=minimal north-last

at the additional gains they realized over their predecessor and the tradeoffs experienced as compared to their predecessors.

Virtual Cut-through flow control allows flits to "cut-through" to the next hop without waiting for the entire packet to arrive. Hence, they reduce latency. However, the trade-off is that the switch now needs to be allocated on a flit-by-flit basis.

Wormhole flow control improves upon virtual cut-through by drastically reducing the number and size of buffers needed, since a blocked packet no longer needs to be buffered in its entirety. The trade-off though is the poorer throughput due to upstream channels being held when packets are blocked, thus causing more blocking than VCT. Instead of sending a credit for each packet buffer, now, a flit buffer being released triggers the sending of credits.

Virtual-channel flow control improves the throughput of wormhole flow control by alleviating blocking with virtual lanes overlayed on physical channels. It also allows deadlock-free networks to be built, and can be used to provide Quality-of-Service. This comes with a slight increase in complexity of the switch (more number of ports) or the allocator (more requests and responses). There is also some overhead incurred as data flits and credits need to encode the virtual channel they belong to. This is a rather small overhead, with 2 virtual channels, it is 1 bit per flit, and with 4 virtual channels, it is 2 bits per flit.

Flit-reservation flow control extends the throughput of virtual-channel flow control through its advance scheduling and just-in-time buffering. Conversely, it halves the number of buffers required to achieve a certain level of throughput. If fast control wires are available, flit-reservation flow control also eliminates routing latency. The trade-off is increased complexity, since now, instead of making channel and buffer allocations based on the current information, allocations are made by scanning the reservation table. There is also additional overhead on the control flits since they have to carry the arrival time stamps of their data flits.

	+	-
SAF		
VCT	Reduction in latency	Allocate switch on a flit-by-flit basis
WH	Reduction in #buffers	Poorer throughput Flit-sized buffer credits
VC	Increases throughput Avoids deadlock Quality-of-service	Increased complexity of switch or allocator Overhead in data flits and credits to identify vcs (1 bit-2 VCS, 2 bits-4 VCS)
FR	Increases throughput OR half #buffers Eliminates routing latency if fast wires are available	Increased complexity (scan rsvn table) Overhead in control flits and credits to store arrival times

A high-level overview of flit-reservation flow control was presented, explaining the two cases experimented with: fast control versus leading control. It was found that even with the control wires having the same speed as the data wires, and control flits just leading the data flits by 1 cycle (which is the usual way for all other flow control techniques), throughput is still improved to that achievable by virtual channels with twice the number of buffers. The reason being that the dedicated control wires are a lot less contended than the data wires. Hence, while data flits get blocked, control flits get further ahead and is better able to schedule buffers ahead of time and use them efficiently.

3.3 Topology

3.3.1 Torus

The basic formulas which have been used for the torus are explained to make sure everybody understood the concepts behind these formulas.

W1, the pin-limited bandwidth = $W_n/2n$
W2, the bisection-limited bandwidth =W_s/B_B, where
W_s is the wire limitation across the bisection, examples being the maximum number of wires that can cross the backplane, or the number of tracks across a chip for on-chip interconnects;
B_B is the number of channels across the bisection = $2N/k$, which is the number of rows in a torus * 2 (since each row is a ring)
Thr = [W.(8/k)]f, because

$$\Theta = \frac{b}{\gamma_{max}}$$

and b = fW, while γ_{max} = k/8

γ_{max} of a torus for uniform traffic is k/8 because it is the average number of hops / number of channels contributed by each node = $(nk/4)/2n$.

For a mixed-radix torus, the channels on the dimension with the most number of nodes are those experiencing the highest load. Hence, γ_{max} is the load on these channels. For a torus with 8 nodes along the X dimension and 4 nodes along the Y dimension, the X dimension is the one more heavily loaded, and hence, γ_{max} is 1.

3.3.2 Butterfly

For uniform traffic, γ_{max} for a switch has been presented in class as 1. An easy way to see why this is so is to look at a 4 by 4 switch. Since each input of the switch has an equally likely chance to go to each of the 4 outputs, each input contributes a load of 1/4 to the output channels. Thus, each output channel is loaded to 1/4*4 = 1 and γ_{max} is 1.

Figures 9 and 10 of the *Benes* networks were also shown, to illustrate their non-blocking nature. Note that non-blocking networks, like crossbars, allow simultaneous access of different output ports by different input ports.

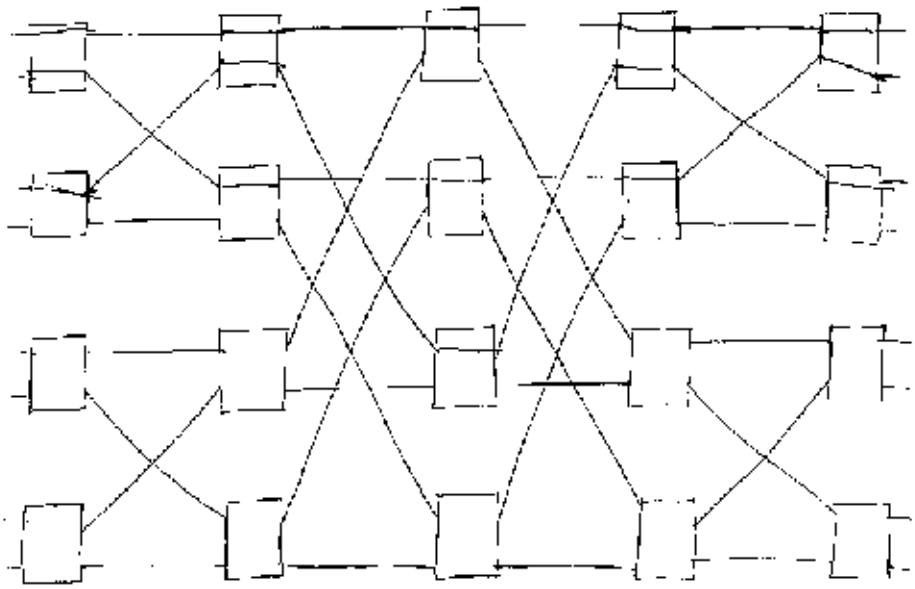

Figure 9: A benes network with 8 input and output ports. Traffic routing each input port to the output port with the same numbering is shown to illustrate the non-blocking nature of the Benes network.

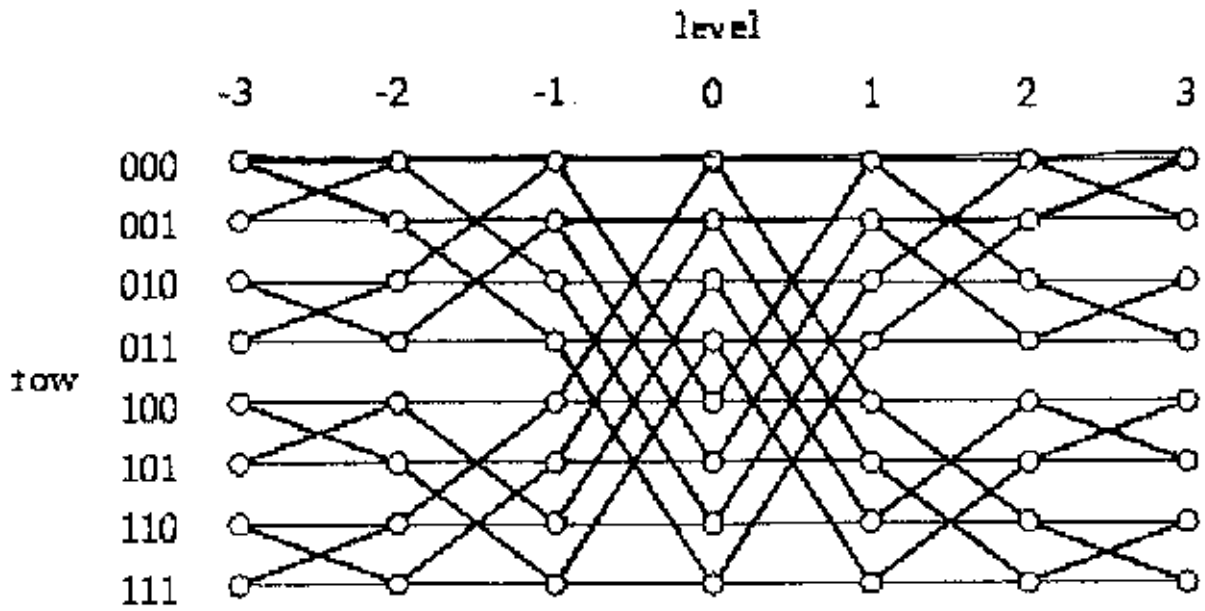

Figure 10: A benes network with 16 input and output ports.

3.3.3 Channel Slicing

Channel slicing is where all the flits of a wide channel is squeezed into a narrow channel, thus increasing serialization delay.