A Communication Model Based on an n-Dimensional Torus Architecture Using Deadlock-Free Wormhole Routing

Philip Hölzenspies, Erik Schepers, Wouter Bach, Mischa Jonker, Bart Sikkes,
Gerard Smit and Paul Havinga
Departement of Computer Science of the University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
{p.k.f.holzenspies, e.m.schepers, w.f.bach, m.d.s.x.jonker, b.sikkes}@student.utwente.nl
{smit,havinga}@cs.utwente.nl

Abstract

Routing on a two-dimensional torus architecture by means of the wormhole routing algorithm is introduced and extended to an n-dimensional torus model. To prevent blocking deadlocks caused by this algorithm, a multiple virtual channel solution is introduced. An implementation of virtual channels is introduced that allows channels with higher labels to pre-empt 'lower' channels. This algorithm is tested with a simplified model of a HiperLAN/2 receiver. The model proves to be capable of running this application on the Chameleon [1] architecture.

1. Introduction

With the growth of the market for hand-held devices, previously second rate criteria gain priority. Energy consumption and heat dissipation become vital parameters in hardware design.

ASICs meet these criteria best of all, but their inflexible nature, long time-to-market and huge development costs makes them a less-than-ideal alternative. On the other end of the flexibility spectrum lies the alternative of software implementation, which would require Instruction-Set-Processors (ISPs). ISPs, however, have a bulky energy consumption and fail to meet processing power requirements for many (in particular streaming) applications [2].

The golden mean between flexibility, processing power and energy consumption can be found in reconfigurable architecture. As some of the tasks (mostly the control-flow oriented ones) are run on an ISP, the remainder of tasks is loaded into a dynamically reconfigurable architecture (FPFA) [3].

When a task is loaded onto an available FPFA, or a running task is pre-empted, the FPFA is dynamically reconfigured for the new task. Because multiple independent tasks run on a torus of FPFAs simultaneously and due to the dynamic allocation of said tasks, routing can not be implemented in the application code; it must be fixed for the architecture. Hence, reconfigurable architectures call for a generic routing algorithm, fit for streaming algorithms and independent[1] of allocation at run-time.

The current research project in the field of reconfigurable computing at the University of Twente is Chameleon [1]. Chameleon is a System-On-a-Chip (SOC) [4], which, in its final implementation, is to include a StrongARM processor, an FPGA and somewhere around two hundred FPFAs. Typical target applications include HiperLAN [5] and Turbo en- and decoding [6]. Chameleon will be used as a test case for the routing strategy suggested in this paper.

A Chameleon application consists of different functions. In the Chameleon architecture these functions will be mapped on tiles resulting in a dataflow from tile to tile. When several applications are running on the architecture they could get in each other's way. If the allocation of the functions is done at random, this will lead to inefficiencies and deadlocks. To prevent these deadlocks, a routing algorithm has to be designed so that all functions can send data when they have to. This routing algorithm must provide functions with transparent means of communication, i.e. independent functions remain unaware of each other.

2. Theory: Pre-emptive Wormhole Routing on a Torus

2.1. The Basics in Two Dimensions

As one of the possible communication topologies a two-dimensional torus with unidirectional channels (Fig. 1) is

[1] As far as correctness is concerned. Optimization might still require the allocation algorithm to be dependent of the routing algorithm.

Proceedings of the Euromicro Symposium on Digital System Design (DSD'03)
0-7695-2003-0/03 $17.00 © 2003 IEEE

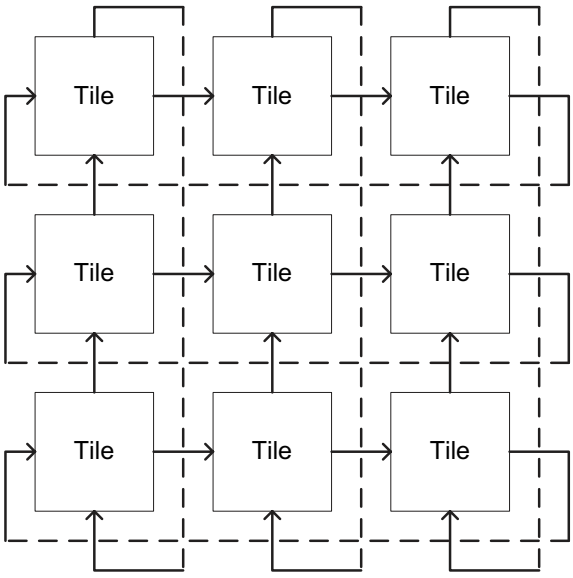

Figure 1. Two-dimensional torus

suggested. The unidirectional constraint is required for, but doesn't suffice to guarantee, dead-lock free routing. The channels depicted here can be implemented as buses. It will become evident later on, that a means of control in the opposite direction is required to provide blocking support.

2.2. Wormhole routing

Wormhole routing is actually not a routing, but a switching technique. The following citation from [7, p. 5] describes the basic principle quite clearly:

> "Wormhole routing uses a cut-through approach to switching. A packet is divided into a number of flits (flow control digits) for transmission. The header flit (or flits) governs the route. As the header advances along the specified route, the remaining flits follow in a pipeline fashion. If the header flit encounters a channel already in use, it is blocked until the channel becomes available. Rather than buffering the remaining flits by removing them from the network channels, as in virtual cut-through, the flow within the network blocks the trailing flits and they remain in flit buffers along the established route."

2.3. Routing algorithm

A minimal, deterministic routing algorithm for meshes known as *XY-routing* [7] is adapted to work deadlock free on a two-dimensional torus[2]. In the XY-routing algorithm a flit stream is pumped to the right, until it reaches the destination column. Next, it is pumped upwards to the destination row.

Using this algorithm, the stream can make - at most - one turn: from a horizontal direction, to a vertical direction. In more general terms, the stream switches its traveling dimension only once, i.e. on a two-dimensional torus. Intuitively, we can expand this model to n dimensions, where a stream may only switch from dimension i to dimension $i+1$ when the proper i^{th} coordinate is reached, making - at most - $n-1$ turns. Since we have unidirectional channels, this method guarantees a shortest-path traversal.

In its nature the XY-routing algorithm is not deadlock free, not even when only considering one dimension. Deadlocks occur when flit stream φ is blocked by flit stream ψ, while ψ is destined for one of the processors already pumping flit stream φ. An example to clarify (Fig. 2):

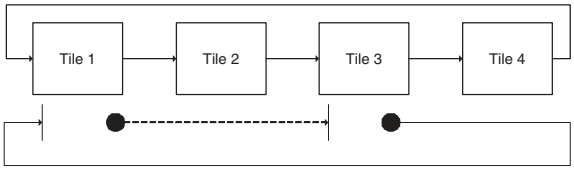

Figure 2. Deadlock situation

Consider a one-dimensional unidirectional torus (i.e. a unidirectional ring) with four tiles. Now, suppose the following conditions hold:

- Tile 1 wants to send a flit stream to tile 4
- Tile 3 wants to send a flit stream to tile 2
- Tiles 1 and 3 start transmission simultaneously

The stream from tile 1 will not be able to proceed past tile 2 and the stream from tile 3 won't go any further then tile 4 because tiles 3 and 1 respectively are unavailable for stream transmission from their neighbours. Hence, deadlock occurs. The cause of the problem lies with the loop in the architecture. To physically eliminate the loop (i.e. to use a mesh) is not an option, since the tiles are connected by unidirectional channels. The solution to this problem is the use of virtual channels [8], whereby a virtual spiral is created (Fig. 3).

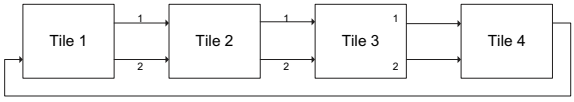

Figure 3. Virtual channels on a one-dimensional torus

[2]This will be expanded to a trivial amount of dimensions later on.

2.3.1. Virtual Channels

Multiple virtual channels are *emulated* on the single physical channel between two tiles. On a one-dimensional torus two virtual channels need to be introduced (labeled 1 and 2). When initiated, a flit stream will always be pumped through channel 1 and only when it crosses the 'edge' of the torus (i.e. when it started at a tile, other than 0 and it arrives at the receiving end of tile 0) will it shift to channel 2.

Extension of the strategy to n dimensions requires the introduction of new virtual channels. Another example to illustrate:

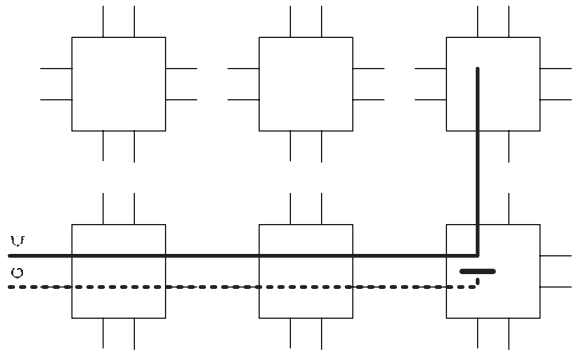

Figure 4. Two-dimensional torus

Consider a two-dimensional torus with two virtual channels in both directions (Fig. 4). When stream φ, being pumped on channel 1, turns to the second dimension, it will also be assigned to channel 1. Now stream ψ, being pumped on channel 2, catches up with φ and turns at the same tile. It can't be assigned to channel 1, since it is already occupied. If it would be assigned to channel 2, a stream starting at a higher tile that traverses the torus' edge would be blocked by ψ and might, itself, block ψ, hence causing a deadlock. The problem can be solved by adding a third channel in the second dimension only (Fig. 5).

The example above can be generalized for an n-dimensional torus. When a stream φ turns from dimension $i-1$ to dimension i, it preserves its channel. Since a stream ψ traversing the edge of dimension i must be able to pass stream φ, there must always be a channel with a higher label than the channel through which φ is being pumped. Say dimension $i-1$ had j channels, i must have at least $j+1$. The proof is by induction.

The proof of the deadlock-free property will be formalized here:

Lemma 2.1 *No cycles in the dependency graph $\Leftrightarrow$ deadlock-free*

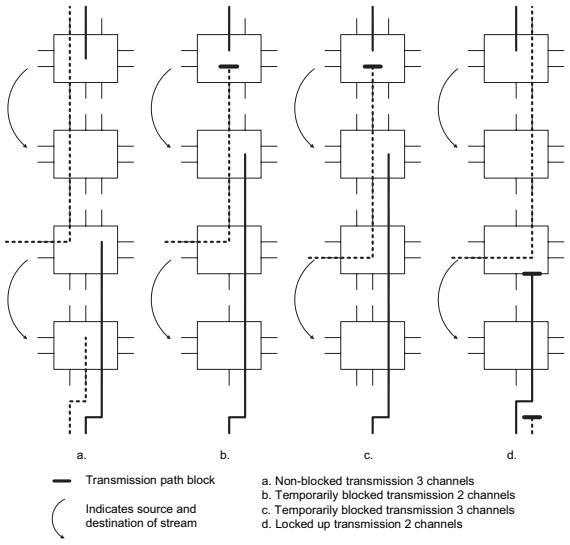

Figure 5. Third channel prevents deadlock

Theorem 2.2 *On an n-dimensional torus, where in every dimension nodes can only route in one direction, $n+1$ channels are required to guarantee deadlock-free routing.*

Proof By induction in 3 steps:

1. In a conventional 1-dimensional torus, the dependancy graph contains a cycle. An extra channel is added, while stated that this second channel may only be used when a stream goes over the edge. The cycle is then eliminated and the network is deadlock-free according to lemma 2.1. So, for a 1-dimensional torus, 2 channels are sufficient for the network to be deadlock-free.

2. Assume that n channels are required to eliminate all cycles in the dependancy graph for dimension 1 to $n-1$.

3. When using n channels, cycles in the dependency graph can only be introduced in dimension n and higher dimensions. In dimension n there are n channels available. By adding another channel to this dimension, it can be made free of cycles as well. So for an n-dimensional torus, using the stated routing algorithm, $n+1$ channels are enough to eliminate cycles in the dependancy graph, and thus guaranteeing deadlock-free routing.

2.3.2. Pre-emption

Multiplexing the virtual channels in time is inadvisable; in the case of average to good allocations, only one or (on large n networks) a few channels will be used simultaneously.

All time-slots for unused channels will be wasted. A good alternative would be to use channel pre-emption.

When a stream on channel i runs into a tile that is pumping a stream on channel $i-1$, it will pre-empt that stream and continue right on through. Only after it is finished, the stream on channel $i-1$ will continue. Pre-emption can occur from any direction, except from the processor on the tile (to lower complexity).

The problem with this method, is that it introduces a new chance of deadlock. Streams can pre-empt each other in a circular fashion, therefor cutting eachothers data supply lines, causing neither to finish. This deadlock (which is significantly different from the blocking deadlock in plain wormhole routing) can be prevented by the allocation algorithm that allocates nodes to processes.

3. Practice: Pre-emption capable routers

A possible pseudo-implementation will be suggested to illustrate the problems and solutions in the design of routers for a two dimensional torus.

3.1. Wiring

The routing algorithm described above is used for communication between tiles and for the communication on a tile; between the router and the processor. Figure 6 shows the wiring required on a tile. It shows the buses (marked with a B) and control lines (marked with a C) to and from neighbouring tiles. The buses require, at least, in this implementation, a width of $W+3$, where W is the word size of the architecture. The three extra bits are used for the bus-control. The control lines can suffice with just a single line. This is explained in the section about the protocol later on.

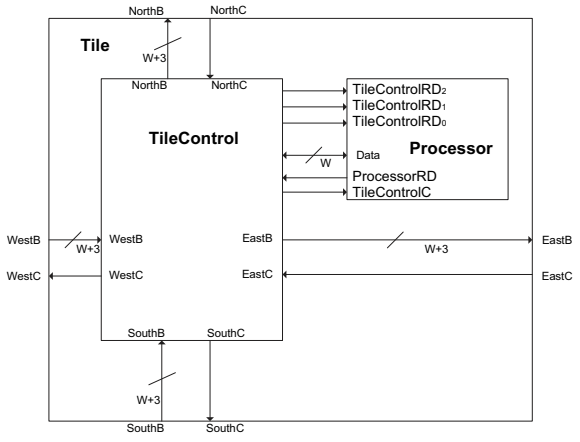

Figure 6. Wiring on a tile

The router and processor are connected through a bi-directional bus, because a double bus is rather expensive. The control signals of the bus can not be implemented bi-directionally, therefore the router-processor bus is only W lines wide and the required control information is implemented separately as mono-directional lines.

3.2. Protocol

A flit stream consists of two header flits and a number of data flits. The first header flit contains the target row and column indexes, the second the total number of data flits still to come (Fig. 7).

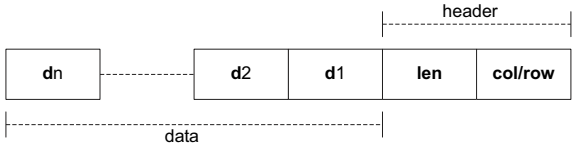

Figure 7. Flit stream

The data bus, as stated earlier, is 3 bits wider than the word size of the architecture. These are required to provide signaling capability. The signals defined on the data bus are shown in Tab. 1.

Signal	C_2	C_1	C_0
Rq_1	0	0	1
Rq_2	0	1	0
Rq_3	0	1	1
Dat	1	-	-

Table 1. Bus layout

Note that when all control lines are 0 nothing is coming in. This is an important detail, because the router should (when expecting data) not keep on decrementing its flit counter if no data is coming in. This may occur when a tile a few tiles down the stream is pre-empted. They will not pass on any signals and hence no signal, means that pre-emption is going on somewhere.

The reason that the processor-router connection is different from the router-router connection, is that these control signals would collide when router and processor request each other's send permissions at the same time, or when the processor is pumping a stream and a pre-empting stream targeting this processor comes in at the router. This problem can be solved by letting the router use all the normal in-bus signal lines to the processor and giving the processor its own control line. Note that the processor has no say in which channel it wants to send on, so a single line suffices; the first signal signifies a request to send and every next signal signifies there's data on the bus. Figure 8 suggests a possible guard to protect the bi-directional bus.

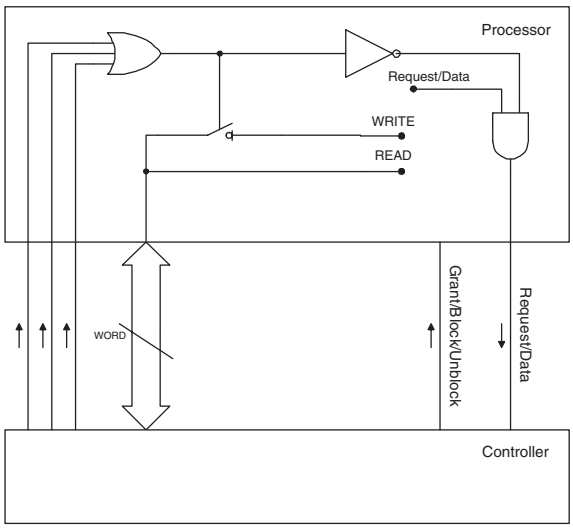

Figure 8. Bus guard

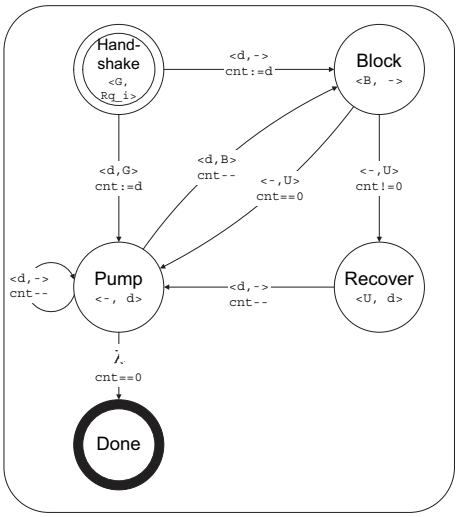

Figure 10. Pump state diagram

3.3. Router States

Figure 9 describes how higher priority channels can pre-empt lower priority channels. It is clearly visible that a Request for channel three is possible in any of the states that do not already require pumping of channel three and will pre-empt said other state in every situation, where channel two requests will only interrupt channel one states or the idle state. The state diagram described here is nested; for every (non-idle) state in Fig. 9 there is a pumping process. This class of processes is depicted in Fig. 10. The DONE transitions are used when the lower layer state diagram (i.e. the pumping process executed by the state) is finished.

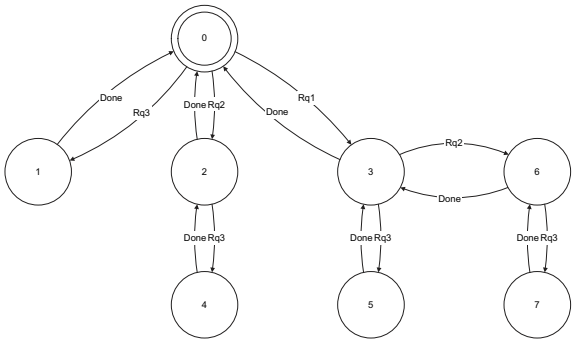

Figure 9. Pre-emption state diagram

The labels marking the transitions describe the input of the router. In Fig. 9 these inputs consist of Request signals. These signals originate from either the tiles on the west or south, or from the processor within the tile. The signals in the lower layer state machine consist of the incoming data bus and the incoming control line, respectively.

Every state in the Fig. 10 also has output, which is depicted as labels inside the states. These labels consist of, at first, the outgoing[3] control line and after that the outgoing data bus.

The state machine in Fig. 10 starts with the handshake between the router and the tile or processor that sent the request whereby the router ended up in this state. Therefore, the output of this state to the requesting node is the *Grant* signal. The request itself is also forwarded directly to the next router or processor in the path, possibly with an incremented channel label, if traversing an edge.

If the receiving end of said request responds with a *Grant* signal on the next clock cycle, pumping of the stream can begin[4]. If not, the router is blocked. Pumping is done in the Pump state, decrementing the flit counter for every incoming data flit, until either a *Block* signal is received or the flit counter reaches 0.

In the event that a *Block* signal is received, the router goes into the Block state, where it sends a *Block* signal to the sending router or processor, until an *Unblock* signal is received. When an *Unblock* signal is received, there are two possibilities: the stream either has more than one flit to go, or it's down to its last flit. When the stream is down to its last flit, no further recovery is required. In case that there is more data left, the tile (or processor) that is sending data to the router has to be given the *Unblock* signal. This is because the tile (or processor) is still blocked at that moment. The unblocking of the previous tile is done in the Recover state. After unblocking the previous tile, we can resume pumping.

[3] To the sending entity, be it neighbour or processor.

[4] Note that the first flit received after the request is the length of the remainder of the stream, so the flit counter is initialized to it

4. Simulation

4.1. Simulation Setup

To test the routing algorithm a simplified model of a HiperLAN/2 receiver has been used, largely due to the fact that an almost complete implementation of this receiver has been made for the Chameleon chip. Therefore, the amount of clock cycles the used functions would take are known. It was also clear that each function would fit on a single tile, which greatly simplifies the simulation model.

The simulation abstracts away from the actual implementation of the receiver functions. The HiperLAN/2 receiver essentially consists of 6 functions. Other parts were ignored because of their lesser relevance to inter-tile communication.

This model of the HiperLAN/2 receiver was implemented in the OMNeT++ [9] environment that was used for simulating the described routing algorithm. OMNeT++ is a discrete event simulation system based on the C++ programming language. A p by p torus of tiles was implemented, where each tile contained a router (a.k.a. tile controller) and a processor. During simulation packets were sent according to how they would be in an implemented HiperLAN/2 receiver. Each processor waited as many clock cycles as it would have taken to calculate its results in the receiver.

The simulation was run for a number of allocation, including best-case, worst-case and random scenarios.

4.2. Results

The initial simulation - without pre-emption - showed that the algorithm is able to route the communication as described in the theory. No blocking deadlocks occured and every flit of data eventually reached its destination. Various cases that did result in a deadlock with normal XY based wormhole routing were fed to the simulator. All of these simulations did result in succesful transmission of all the streams. However, the necessity of testing remains as no specific statistical results that could be used to determine the efficiency of this communication scheme, are yet available. When pre-emption was introduced, a new class of deadlocks was found. These deadlocks should be prevented when scheduling.

5. Conclusions

By combining existing communication schemes, such as wormhole routing and XY-routing, a new communication mechanism is introduced that can be used for deadlock free transmission between tiles in an n-dimensional torus architecture. A specific implementation has been given for the two-dimensional variant. While the algorithm is tested against various deadlock-prone situations and proved to be correct, more simulations still need to be run. The reason for this is that performance figures of this model are yet to be determined.

The implementation of the virtual channels as suggested is flawed, in that it might re-introduce deadlocks. Time-multiplexing isn't an acceptable alternative because of the cost factor.

6. Future Work

More metrics must be gathered to put a number on the algorithm's performancs. This paper does not compare the algorithm (performance wise) with other algorithms currently deployed in the field. Such a comparison is vital for feasability testing.

Research could be done in better routing techniques. One change lies in using other techniques to prevent the deadlocking. A way different from the way channels are used now is channel promotion. Channels that have been waiting longer then get a higher priority over channels that have not been waiting that long.

An allocation algorithm that is aware of this communication algorithm is likely to be a good investment for future implementations. This algorithm should work towards preventing pre-emption, since it is starvation and deadlock prone, if any realtime characteristics need to be guaranteed.

Finally, it should be researched whether the sending frequency and stream sizes of an application can be known at allocation time. This could enable realtime schedulers to be used to streamline communication. To see whether periodic delays to realise generated schedules could be implemented in the routers, further research is required as well.

References

[1] `http://chameleon.ctit.utwente.nl/`

[2] J-Y. Mignolet, S. Vernalde, D. Verkest, R. Lauwereins: "Enabling hardware-software multitasking on a reconfigurable computing platform for networked portable multimedia appliances", ERSA2002, Las Vegas, June 2002

[3] Paul M. Heysters, Henri Bouma, Jaap Smit, Gerard J.M. Smit and Paul J.M. Havinga: "Reconfigurable System Design: The Control Part", PROGRESS2001, Veldhoven, October 2001

[4] IBM Microelectronics Division: "System-on-a-Chip Design: A Time to Market Challenge", 1999

[5] http://www.etsi.org/frameset/home.htm?/technicalactiv/Hiperlan/hiperlan2.htm/

[6] C. Berrou, A. Glavieux, and P. Thitimajshima: "Near Shannon limit error-correcting coding and decoding: Turbo codes", In IEEE Proceedings of ICC 93, pages 10641070, May 1993

[7] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks", Computer, vol. 26, no. 2, pp. 62–76, Feb. 1993

[8] Z. Liu, J. Duato, and L.-E. Thorelli: "Grouping virtual channels for deadlock-free adaptive wormhole routing", in Proceedings 5th Conference of Parallel Architectures and Languages 13 Europe,volume 694 of Lecture Notes in Computer Science, pages 254265. SpringerVerlag, 1993

[9] http://www.hit.bme.hu/phd/vargaa/omnetpp.htm/