Self-optimized Routing in a Network-on-a-Chip

Wolfgang Trumler, Sebastian Schlingmann, Theo Ungerer, Jun Ho Bahn, Nader Bagherzadeh

Abstract Many-cores are on the cusp of becoming state-of-the-art processor technology for the next decade. To guarantee efficient communication between multiple cores, a Network-on-a-Chip (NoC) is considered as an alternative to overcome the limitations of the ubiquitous bus technology.

In this paper, we present an approach to further improve the routing in an NoC with a self-optimized routing strategy. We extended the routers of a network to measure their load and to send an appropriate load information to their direct neighbors. The load information is used to decide in which direction a packet should be routed to avoid hot-spots. Evaluation results show a significant increase in the network throughput. With the self-optimized routing, the NoC is capable of routing up to two times more packets compared to the original routing algorithm proposed by Lee and Bagherzadeh, 2006.

1 Introduction

In 2007 Intel announced a prototype 80-core tera-scale processor [11] using Network-on-a-Chip (NoC) [7] technology. NoC is used as an alternative to the ubiquitous bus technology in order to facilitate communication among many cores. As the process technology shrinks and more cores are integrated on the same chip, the current bus approach for communication among cores will not be sufficient and a technology such as NoC is needed.

Wolfgang Trumler, Sebastian Schlingmann, Theo Ungerer
Department of Computer Science, University of Augsburg, Eichleitnerstr. 30, 86159 Augsburg, Germany, e-mail: {trumler, schlingmann, ungerer}@informatik.uni-augsburg.de

Jun Ho Bahn, Nader Bagherzadeh
Department of Electrical Engineering and Computer Science, University of California, Irvine, California, USA, e-mail: {jbahn, nader}@uci.edu

In an NoC system, processor cores exchange messages using a network as transportation system that is constructed from multiple point-to-point data links interconnected by routers such that messages can be relayed from any source module to any destination module over several links by making routing decisions at the local routers. NoCs apply message passing communication networks similar to massively parallel systems. For NoCs, the advantage of a low latency communication, compared to the off chip communication on high-speed channels among processors, offers new possibilities but also new challenges for the routing in such networks.

There is only limited room for improvements to the topology of an NoC compared to the 2D-mesh due to space and energy constraints on a chip. Therefore, most NoCs employ a simple 2D-mesh for communication infrastructure. On the other hand, significant efforts have been spent on the optimization of routing for NoCs concerning both, the overall throughput and the average latency.

The O1TURN algorithm [18] for example has a provable near-optimal worst case throughput. Another routing algorithm with good performance and deadlock free routing is ROMM [14].

In this paper, we present an approach to increase the network throughput and to lower the average latency based on the local load information of the nodes. The nodes exchange their local load values with their neighboring nodes, which route incoming packets based on this information. The basic idea of this self-organizing, adaptive routing algorithm is inspired by the self-optimization algorithm [20] for load balancing in large scale networks, which is based on the notion of the human hormone system. The underlying architecture [13] for our algorithm, which does not rely on virtual channels, has been developed at the University of California in Irvine.

The artificial hormone system described in [20] piggy backs load information on the outgoing messages. This information is extracted by the receiving node, which decides wether to transfer load, in form of a service, to the origin of the message. In this scenario the communication was constrained by the uniquely identified communication partners of each service. We assumed that one service will not send the message to all other services, but only one of its communication partners at a time. This simulates the way information flows in object oriented software, where one object can call methods of only a few other objects. Furthermore, it is similar to the way hormones distribute their information to only those parts of the tissue which have receptors for these specific hormones and thus can act on this information.

The communication pattern of the self-optimizing algorithm in this paper is more related to the way information is distributed known from the process of morphogenese [21], first described by Turing in 1952. Turing mathematically described his idea of messengers that diffuse into neighboring regions in the tissue of animals and plants, used to organize the creation of regular structures. The concentrations of different messengers are responsible for the creation of the patterns of a zebras or the regular leaf structure of woodruff for example.

The communication pattern of the NoC routers does not change over time nor does the structure or the layout of the routers on the die change. In this sense the self-optimization is more related to the findings of Turing. On the other hand, the simple approach of local load values, without the complicated differential equations of the morphogeneses, is more related to the artificial hormone system as desribed in [20].

The remainder of this paper is structured as follows. The next section describes the architecture of the underlaying hardware. In Section 3, the calculation of the local load is described and the routing algorithm is explained in detail. Simulation results are presented in Section 4 and related work is discussed in Section 5. The paper closes with a conclusion and the description of future work in Section 6.

2 Basic router architecture

The network topology of the NoC is a 2D-mesh with *NxM* routers. Figure 1 shows the architecture of a single router. The router consists of three subsections, the left, the right, and the internal router.

The task of the internal router is to inject packets into the network. Packets that arrived at their destination are also ejected by the internal router. The left router is used to route packets to the left (west) and the right router routes packets to the right (east), respectively. Both routers can route packets to the north as well as to the south but there is no connection between the east and west direction. This assumption divides the network into two separate networks where packets can go either to the east or the west of a router, but there is no turning back in the horizontal routing direction. This architectural approach guarantees the NoC to be dead-lock free [5].

Clock boosting was introduced to improve the performance of the NoC. A packet consisting of multiple flits (flow control digits) can be transmitted at different clock speeds. With clock boosting only the routing decision for the head flit is done at normal clock speed. After the route for the packet is chosen, the body flits can be routed with an increased clock speed. By multiplying the clock frequency more than one body flit can be routed during a normal clock cycle. The clock boosting can double or quadruple the basic clock frequency, allowing two or four body flits to be routed at the same time.

The routing decision of a router is straight forward using a clockwise priority scheme to select the packets for routing. Starting from the north (top) input channel the router examines the head flit from the buffer and tries to set the route for this packet if possible. If there is no head flit in front of the buffer or if the route can not be set because the output channel is already occupied by another packet, the router picks the next buffer in a clockwise order and repeats the aforementioned steps. More details about the clock boosting and the routing decision can be found in [13].

The internal FSM of the router is easy to implement but does not take the current load situation of the network into account. Our approach is to improve the routing decision by considering the current load of the neighboring routers, finding routes to avoid hot-spots. Furthermore, when the original algorithm stops routing due to a congested network, our algorithm can use alternative routes, bypassing the heavily loaded routers, which leads to a higher overall network throughput.

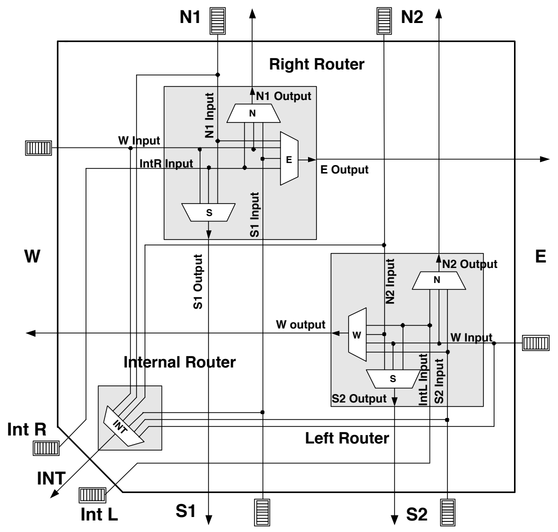

Fig. 1 Architecture of a router

3 Routing Algorithm

Every router performs a routing decision whenever a packet arrives. The routing decision consists of two consecutive steps. First, a *routing function* creates a set of possible output channels (next destinations) for a packet. Afterwards, the *selection function* calculates quality values for all possible routes and selects the most appropriate one for the routing of the packet.

In the current setup, the routing function creates a set with all nodes that lead to the desired direction of the packet. For packets going from west to east (right direction) the set may contain the north, west, and south channels. For packets going from east to west (left direction) possible output channels can be north, east, and south, respectively. If a packet has to be routed in south or north direction only, this desired direction is used for the routing.

The local load of a router is propagated to its neighbors so they can decide if the router is a good choice to route packet. The local load is not used for the local routing decisions but it is crucial for the routing decision of the neighboring routers. The propagation of the load values is described in Section 3.2.

3.1 Self-Optimization-Algorithm

3.1.1 Selection Function

The Self-Optimization-algorithm calculates the quality of a possible route based on Equation 1. To yield a load value in the range of 0 to 100 all parameter values are normalized.

$$quality = direction - \%of\,remaining\,flits - 4 * pload \tag{1}$$

The first value (*direction*) is a sort of bonus if the route leads to the destination of the packet. A value of 200 is added for a route heading towards the destination and 0 for a route which leads the packet farther from its target.

The value *%of remaining flits* expresses the percentage of a packet that must still be sent on the selected output channel. If there is currently a route set for a packet, the amount of the remaining flits of the packet is divided by the packet length. This value is used to penalize a route, if there is already a packet on that route. The amount of remaining flits is used to give an idea of how long the route will be occupied.

The *pload* value is the load value of the neighboring router in the desired direction. The calculation of the *pload* value is described in Section 3.2. The propagated load of the considered direction (*pload*) is multiplied by four and subtracted from the other values. The higher the load of the neighboring routers, the less attractive is the route in that direction.

3.1.2 Calculating the Load of a Router

The SO-algorithm tries to avoid hot-spots by spreading the offered load as good as possible to the available routers as long as free capacity is available. The load value, which is also in the range from 0 to 100, is used for the calculation of a channel's quality. The load calculation of the SO-algorithm uses the utilization of the buffers as a degree for the load of a router. The more the buffers are filled with flits, the higher is the load of a router. Therefore, the load can be calculated by the fraction of currently available flits for the available buffer size.

$$load = \frac{local}{maxload} * 100 \tag{2}$$

The value *local* is the sum of the flits in the input buffers of a router. The amount of flits from injection buffers are limited by the capacity of an input buffer, because the injection buffers are assumed to be unlimited for simulation purposes. The maximum utilization of a router, *maxload*, is reached if all input buffers are completely filled with flits.

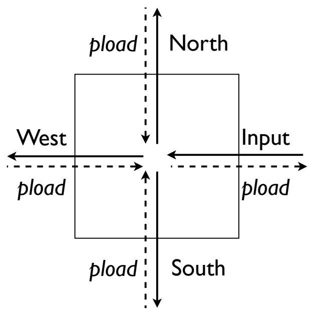

Fig. 2 Propagation scheme for the left router

3.2 Load propagation

To propagate the local load, a router sends the load value to its neighboring routers. Therefore, the overall load of a router is calculated and then propagated to the neighbors.

The propagated load value, *pload*, not only includes the local load, but also the load values of the surrounding routers. The information of the surrounding routers is used to increase or decrease the local load of a router. This guarantees that the load information of a heavily loaded router is not only sent to its direct neighbors but is spread from the center of a possible hot-spot to the surrounding parts of the network. The closer a router is located to a hot-spot the higher is the propagated load value, which is used to decide if a packet should be routed to this node or not.

The calculation of the value *pload* is shown in Equation 3. Two-thirds of *pload* are taken from the local load value *load*. One third of the propagated load depends on the load of the neighboring routers.

$$pload = \frac{1}{3}\left(2 * load + \frac{\sum pload(dir)}{|pload(dir)|}\right) \tag{3}$$

The load that is propagated into one direction depends on the load of the routers from the possible direction that can be used to route a packet. For example, the load of the left router (see Figure 2) , the router that can route either to the north, south, or west, takes the load values from these three neighboring routers and sends this information to the right (east) router. Depending on the direction, the *pload* value incorporates the load value of up to three neighboring routers. For the *pload* value propagated to the south of the right router only the load information from the north and west router is taken into account, because the routers can not route a packet back to the direction where it came from. Thus, the only possible routes are to the north and west.

4 Evaluations

We conducted extensive experiments with different network sizes ranging from 4x4 up to 16x16. Next, we will describe the traffic patterns used for the generation of the network load and discuss the results for a 4x4 and 8x8 2D-mesh network in comparison to the results of the design from the University of California in Irvine. Afterwards, we compare the performance gain of the network in terms of increased network throughput.

4.1 Traffic generation

We evaluated our algorithms with four different traffic patterns as proposed in [7] (Chapter 9). We used the same four traffic patterns as they did at the UCI to have a basis for a direct comparison of the results.

The traffic patterns used are *matrix transpose*, *bit reverse*, *bit complement*, and *uniform random*. The target address of a packet is generated out of the source address of a node by applying one of the aforementioned traffic patterns. For these simulations, the width and height of the 2D-mesh network is assumed to be a power of two. Without this assumption some of the traffic patterns might produce invalid destination addresses.

For every simulation setup, we conducted multiple runs and calculated the average network latency to minimize the impact of possibly good circumstances in one simulation run. The network had a warm-up phase of 1000 cycles at the beginning of every simulation. The measurement was done during 100,000 cycles following the warm-up phase. The simulation stopped if the average delay of the packets exceeded a threshold of 200 cycles.

The injection of the flits was chosen to be the worst case, which means, that all nodes inject their flits at the same time resulting in a high network load. Assuming that the nodes would distribute the flit injection over time, the results are even better than shown in the charts.

The packet injection rate can be calculated from the injection rate of the flits. In our simulations, we used a fixed packet length of nine flits (one head flit and eight body flits). Therefore, the packet injection rate can be calculated by dividing the flit injection rate by nine.

4.2 4x4 NoC

The first simulations were done on a 4x4 2D-mesh with the aforementioned traffic patterns. The results are shown in Figure 3. The charts show the generated traffic on the x-axis and the measured average latency on the y-axis. The generated traffic is given in flits per node per cycle which is a value ranging from 0.05 to 1. If the generated traffic is 1, a new flit is injected into the internal buffer of the routers at every node in every cycle. If the traffic is below 1, there are a few cycles delay between the injection of new flits, e.g. at 0.5 a new flit is injected every second cycle.

For a direct comparison of the results from Irvine and our results, we plotted all the data into one chart. Every chart shows the results with 1x, 2x, and 4x clock boosting for both.

Concerning the matrix transpose traffic pattern there is hardly any improvement of the SO-algorithm compared with the original algorithm. Both can route all packets with a nearly constant delay up to the maximum load, if the clock boosting is used. The saturation point of the network seems to be about the same for both algorithms.

The bit reverse traffic pattern first shows considerable differences. The saturation point for the SO-algorithm is about 0.65 and about 0.45 for the original algorithm without clock boosting. The maximum throughput of the SO-algorithm is even better for the 2x clock boosting. While the original algorithm begins to saturate at about 0.8, the SO-algorithm routes all packets with nearly the same delay up to the maximum injection rate. The results with 4x clock boosting are again the same for both algorithms.

Bit complement traffic shows interesting results in terms of the algorithms saturation points. The original algorithm has a saturation point of 0.35, 0.5, and 0.85, respectively. The SO-algorithm first increases the latency, which creates a kind of plateau phase. This plateau has a different length for different clock boosting values. The plateau also appears at the 4x clock boosting, but the SO-algorithm can route all the packets with only a slight latency increase up to the maximum injection rate and does not reach the saturation point.

The uniform random traffic pattern seems to be the hardest of all four traffic patterns, because the saturation point of the original algorithm is reached earliest. The SO-algorithm shows very good behavior especially for the 2x and 4x clock boosting. In both cases, the SO-algorithm does not reach the saturation point but can route all packets up to the maximum injection rate. The uniform random traffic pattern shows similar behavior for the SO-algorithm than in the previous traffic pattern. There are two plateaus for the 1x and 2x clock boosting at different injection rates. The first plateau is also visible for the 4x clock boosting but the second one does not appear.

The explanation for the appearance of the plateaus can be derived from the way the SO-algorithm selects the routes for the destination of a packet. As long as there are enough good alternative routes to the destination, the SO-algorithm automatically selects the next best (shortest) path. For example, a packet at the node (1,1) should be routed to (3,2) and if the buffers of the router to the east (2,1) are filled, the SO-algorithm routes the packet to (1,2) instead. If the injection rate increases, more buffers are completely filled and more congestions arise in the network. At the same time, since the quality of the alternative routers are also getting worse, the SO-algorithm selects routes by avoiding both shortest but heavily loaded routes.

If the SO-algorithm starts routing packets not on one of the shortest paths, the average latency increases due to the additional hops and the possible congestions.

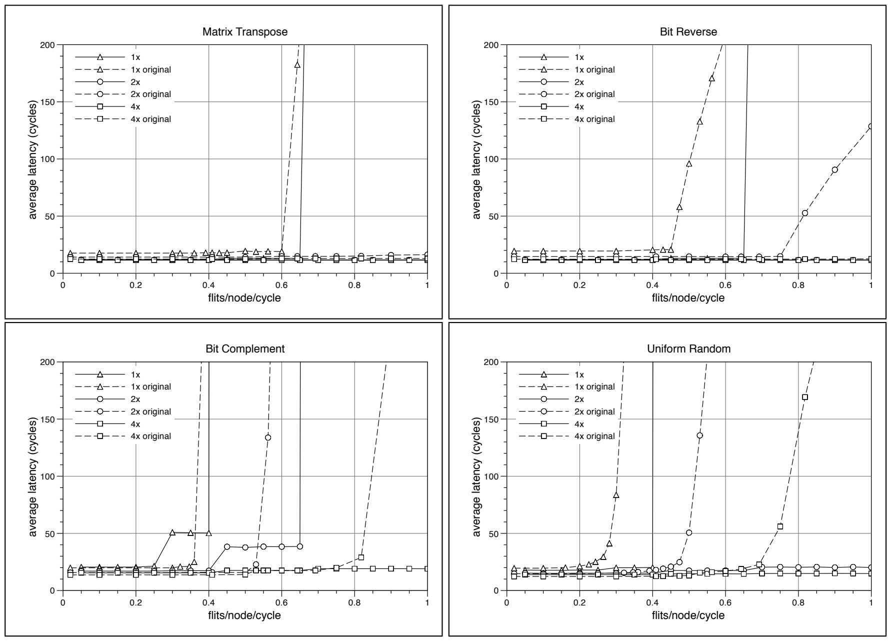

Fig. 3 Simulation results for a 4x4 2D-mesh

On the other hand, the alternative routes around heavily loaded nodes give the SO-algorithm the ability to defer the ultimate saturation point to a much higher injection rate and in most cases it can handle the maximum injection rate.

4.3 8x8 NoC

We also conducted experiments with 8x8 2D-mesh networks and the four traffic patterns to see how the SO-algorithm performs in larger networks.

The amount of injected flits depends on the amount of nodes. Therefore, if the number of nodes is doubled in each dimension, the amount of nodes in the network increases exponentially and so does the amount of flits injected into the network. On the other hand, the exponential growth in the network size might offer more alternative routes for the SO-algorithm to be selected as routes for the packets.

The results of the simulations of an 8x8 2D-mesh are shown in Figure 4 for 1x, 2x, and 4x clock boosting. The original algorithm reaches its saturation point for all traffic pattern at an injection rate considerably below the maximum injection rate.

With the matrix transpose traffic pattern, the SO-algorithm performs much better than the original algorithm. Especially with 4x clock boosting, where the original algorithm saturates at about 0.65, the SO-algorithm can route packets up to the maximum injection rate with a low average latency. The same applies for the bit

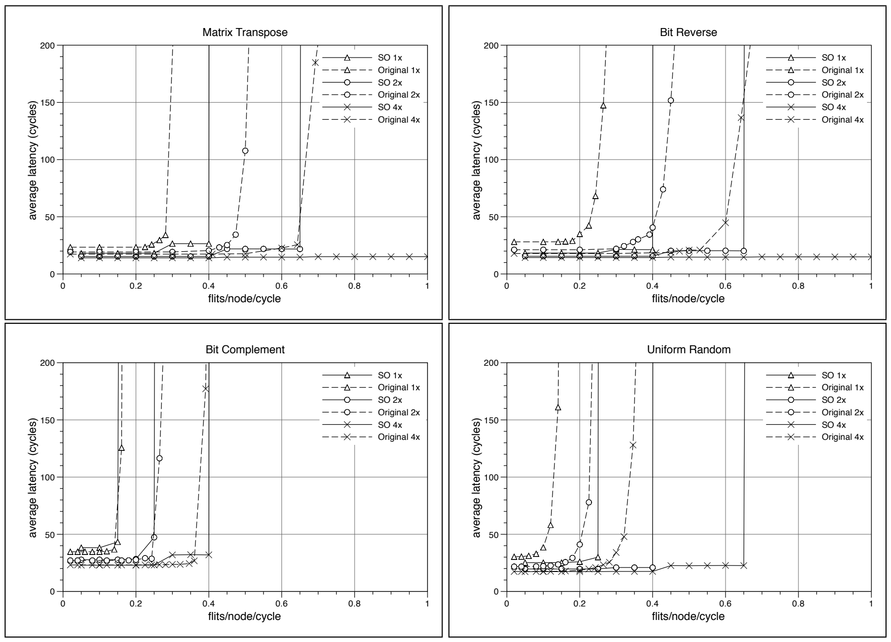

Fig. 4 Simulation results for an 8x8 2D-mesh

reverse traffic pattern where the original algorithm performs even worse while the SO-algorithm performs slightly better than with the matrix transpose traffic pattern.

The bit complement seems to be challenging for both, the original and the SO-algorithm. In this case the SO-algorithm does hardly outperform the original algorithm, in contrast to the other three traffic patterns. We had similar results for the 4x4 network. One of the influencing factors seems to be the network architecture, which is chosen to avoid dead-locks by design. Because of the two separated networks for the horizontal directions, the SO-algorithm does not offer enough choices for alternative routes. A packet cannot be routed back in horizontal direction in contrast to the vertical direction where a packet can go back and forth if the packet has been routed horizontally, at least once after a vertical transfer.

In the lower right chart of Figure 4 the results of the uniform random traffic pattern are depicted. In this case, the SO-algorithm can handle the double injection rate than the original algorithm before it reaches the saturation point. The difference with the bit complement traffic pattern is due to better distribution of the traffic over the whole network, which seems to be the favored kind of traffic for the SO-algorithm.

The plateaus known from the 4x4 network can be observed at all four charts of 8x8 network. As already mentioned, this behavior occurs when the SO-algorithm starts to use alternative routes, which are not on the shortest path to the destination.

4.4 Performance comparison

To compare the network throughput of the original and the SO-algorithm Tables 1 and 2 show the throughput gain in percentage for the 4x4 and 8x8 network, respectively. The original algorithm's results are taken as base for the calculation of the throughput gain. Therefore, a value of 0% means that both algorithms perform equally and a value of 100% means that the SO-algorithm performs twice as good as the original algorithm.

For an algorithm that reaches its saturation point, an average injection rate latency of 50 cycles was taken for the base calculation. When the original algorithm saturates and the SO-algorithm does not, the throughput gain becomes much higher than the calculated value shown in the tables. This is because in all these cases the average latency of the SO-algorithm is much better than 50 cycles. These values are marked with a * to denote that the original algorithm saturated in contrast to the SO-algorithm.

Table 1 Throughput gain for a 4x4 NoC

Traffic Pattern	1x	2x	4x
Matrix Transpose	13%	0%	0%
Bit Reverse	41%	* 23%	0%
Bit Complement	8%	75%	* 20%
Unifrom Random	37%	* 96%	* 35%

The throughput gain for the 4x4 network ranges from 0% for the matrix transpose to more than 96% for the uniform random traffic pattern. As mentioned before in the case of the uniform random traffic pattern the throughput gain is even higher due to the fact that the original algorithm has a 50 cycles delay for the chosen injection rate while the SO-algorithm has an average delay of about 21 cycles. For the 8x8 2D-mesh network, the throughput gain ranges from 0% for the bit complement up to 127% for the uniform random traffic pattern.

Table 2 Throughput gain for a 8x8 NoC

Traffic Pattern	1x	2x	4x
Matrix Transpose	37%	35%	* 53%
Bit Reverse	73%	58%	* 63%
Bit Complement	7%	0%	8%
Unifrom Random	127%	90%	103%

The tables confirm our initial assumption that the SO-algorithm will perform better for larger networks due to the larger amount of possibles routes. The bit complement is the sole exception to this assumption. Further investigations are needed to better understand the reasons for the poor throughput gain in this case.

5 Related Work

In designing Network-on-a-Chip (NoC) systems, there are several issues to be considered, such as topology, routing algorithm, performance, latency, and complexity. Because of its flexibility, architectures based on NoC are getting more attention. As a feasible topology in NoC systems, the mesh is getting popular for its modularity; it can be easily expanded by adding new nodes and links without any modification of the existing node structure.

Another issue in NoC environment is the routing algorithm. In terms of delivering mechanism, wormhole routing has increasingly been advocated as a method of reducing message routing latency. In wormhole routing, a packet is decomposed into flits or flow control units, and the packet follows through the network one flit after another. On the other hand, in terms of the way of selecting a path among the sets of possible paths from source to destination, the routing algorithms are classified as deterministic/oblivious and adaptive ones [5]. The oblivious/deterministic routing algorithms choose a route without considering any information about the network's present condition, resulting in relatively simple design complexity. Adaptive routing algorithms use the state of the network such as the status of a node or link, the status of buffers for network resources, or history of channel load information. Even though the adaptive routing algorithms utilize the flexibility in routing paths, the hardware design complexity is usually increased. Depending on the degree of adaptivity, minimal adaptive and fully adaptive routing algorithms are refined. DOR (dimension-ordered routing) [19], ROMM [14], and O1TURN [18] are examples of deterministic or oblivious algorithms. Some researchers have developed better performance routing algorithms using adaptive routing algorithms [10, 4, 6, 2, 9, 8]. The SO-algorithm is an adaptive routing algorithm that does not use virtual channels.

The adoption of virtual channel (abbreviated to VC) has been prevailing because of its versatility. By adding virtual channels and proper utilization of their channels, deadlock-freedom can be easily accomplished. Network throughput can be increased by dividing the buffer storage associated with each network channel into several virtual channels. By proper control of virtual channels, network flow control can be easily implemented [3]. Also to increase the fault tolerance in a network, the concept of virtual channel has been utilized [1, 12]. However, in order to maximize its utilization, allocation of virtual channels is a critical issue in designing routing algorithms [22, 16]. Furthermore, the buffers of the virtual channels are very expensive in terms of chip size. The extra chip size needed for the additional logic of our SO-algorithm is negligible compared to the chip size needed for the buffers of virtual channels and the logic for the channel allocation.

A similar approach, using stress values, is described in [15]. The stress values are exchanged between the direct neighbors in the network. With our calculations the load value is not only exchanged with the direct neighbors, but diffuses to the surrounding area of load value's source. Furthermore, there are less choices for our algorithm to chose the best routes due to the two separated networks, which guarantee deadlock free routing by design.

6 Conclusion and Future Work

In this paper we presented a routing algorithm for an Network-on-a-Chip that yields a significant throughput gain for 2D-mesh networks. The SO-algorithm calculates the load of a router based on the amount of flits in the buffers. With this approach the network throughput can be significantly increased. The throughput gain is up to 127% compared to the original algorithm. The throughput gain is highest for the uniform random traffic pattern, which in our opinion is especially relevant for our current research where we will investigate task allocation mechanisms on an NoC. The expected traffic in such a dynamic and steadily changing environment will lead to a traffic pattern comparable to the uniform random.

Based on our latest results, we have plenty of ideas to improve the SO-algorithm. The first will be to combine the SO-algorithm with another approaches that is described in [17]. The idea is to use the SO-algorithm as a base value for the local load and to increase or decrease it by the amount of I/O operations a router can process during a cycle. If the input channels of a router are filled with flits, the router is loaded to a level of 100% (concerning the SO-algorithm) but the situation is aggravated if the router is blocked and cannot perform any read or write operations. On the other hand, if the router is full but can transfer a maximum number of flits, it might be a better choice.

References

1. S. Chalasani and R. V. Boppana. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Trans. Comput.*, 44(7):848–864, 1995.
2. G.-M. Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.
3. W. J. Dally. Virtual-channel flow control. In *17th annual international symposium on Computer Architecture (ISCA '90)*, pages 60–68, New York, NY, USA, 1990. ACM.
4. W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.
5. W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004.
6. J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320–1331, 1993.
7. J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks – An Engineering Approach*. Morgan Kaufmann Publishers, 2003.
8. C. J. Glass and L. M. Ni. Maximally fully adaptive routing in 2D meshes. In *Proceedings of the 1992 International Conference on Parallel Processing*, volume I, Architecture, pages I:101–104, Boca Raton, Florida, 1992. CRC Press.

9. C. J. Glass and L. M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, 1994.
10. J. Hu and R. Marculescu. Dyad - smart routing for network-on-chip. In *41-st Annual Conf. on Design and Automation*, pages 260–263, 2004.
11. Intel Corporation. Intel's teraflops research chip. http://download.intel.com/research/platform/terascale/teraflops/ FINAL_TeraflopsResearchChip_Overview.pdf, November 2007.
12. F. Jipeng Zhou; Lau. Adaptive fault-tolerant wormhole routing with two virtual channels in 2d meshes. *Parallel Architectures, Algorithms and Networks, 2004. Proceedings. 7th International Symposium on*, pages 142–148, 10-12 May 2004.
13. S. E. Lee and N. Bagherzadeh. Increasing the throughput of an adaptive router in network-on-chip (noc). In *3rd International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, Seoul, Korea, Oktober 22-25 2006. ACM.
14. T. Nesson and S. L. Johnsson. ROMM routing on mesh and torus networks. In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'95*, pages 275–287, Santa Barbara, California, 1995.
15. E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, pages 11126–11127, Washington, DC, USA, March 2003. IEEE Computer Society.
16. H. Rezazad, M.; Sarbazi-azad. The effect of virtual channel organization on the performance of interconnection networks. *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8 pp.–, 4-8 April 2005.
17. S. Schlingmann. Selbstoptimierendes routing in einem network-on-a-chip. Master's thesis, University of Augsburg, September 2007.
18. D. Seo, A. Ali, W.-T. Lim, and N. Rafique. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *32nd International Symposium on Computer Architecture, 2005. ISCA '05*, pages 432–443, Madison, Wisconsin USA, 4-8 June 2005.
19. H. Sullivan and T. R. Bashkow. A large scale, homogeneous, fully distributed parallel machine, i. In *ISCA '77: Proceedings of the 4th annual symposium on Computer architecture*, pages 105–117, New York, NY, USA, 1977. ACM.
20. W. Trumler, T. Thiemann, and T. Ungerer. An artificial hormone system for self-organization of networked nodes. In *IFIP Conference on Biologically Inspired Cooperative Computing*, pages 85–94, Santiago de Chile, August 2006. Springer-Verlag.
21. A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, August 1952.
22. A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. Impact of virtual channels and adaptive routing on application performance. *IEEE Trans. Parallel Distrib. Syst.*, 12(2):223–237, 2001.