# Fast and fault tolerant routing algorithm for Network-On-Chip

Varun V., Raghunandan Menon K.., Karthigha Balamurugan
Department of Electronics and Communication Engineering, Amrita School of Engineering, Coimbatore, India
Amrita Vishwa Vidyapeetham, India
varun1995v@hotmail.com, rnmenon95@gmail.com,b_karthigha@cb.amrita.edu

***Abstract* - Given the rapid progress of technology, transistors are being scaled down at an aggressive pace. The digital designs that are built using such entities are thus, more complex, compact and integrated. The same can be said of the communications systems that facilitates interaction between the various components like the IP cores, input ports, first in first out (FIFO) buffers, decoders, a crossbar, a routing table and output ports, allowing them to function synergistically as a whole. One such popular technology is the Network-On-Chip (NoC), which makes inter- elemental communication very efficient. As the dimensions of the buffers and cores reach atomic proportions, they become more vulnerable to faults due to gate oxide breakdown, ageing, electric field variations, negative bias temperature instability [1] and circuit breaks (due to overheating of circuit elements). As such, attention needs to be given to the network technology to keep it robust and fault-tolerant, preserving its efficiency as much as possible even in the face of faults. Keeping this in mind, a routing algorithm with rerouting option is proposed, which covers deadlocks and wire/router faults, thereby raising fault tolerance. The algorithm, named XY-Cruise algorithm, promises to determine the shortest path in a mesh topology based network in the presence of faults and deadlocks, as well as to develop a suitable architecture to realize the algorithm for the proposed rerouting purpose.**

***Index terms – Fault tolerance, networks-on-chip (NoC), reliability, routing algorithms, rerouting***

## I. INTRODUCTION

Continuously shrinking transistor dimensions enable ever-increasing density on modern microchips: each new technology node facilitates additional cores in chip multiprocessors. For instance, the Intel SCC contains 48 cores [2], the Tilera Tile64 has 64 cores [3], and the experimental Intel Polaris chip comparison as many as 80 cores [4]. Some may have over a hundred, as is the case of ASOCS ModemX, which consists of a maximum of 128 cores and the Nvidia Fermi, which comprises 448 cores [5]. High core numbers highlight efficient, faultless and scalable interconnects that are capable of providing communication among the processor cores. However, crossbar interconnects and system bus communication [6] – [9] facilities have not been scaled efficiently enough, as has been done for processors. As the scope and range of microprocessor applications widen, they become more demanding where performance parameters such as speed of data transfer amongst the cores, latency of operations and tolerance to faults – both physical and logical – are concerned. In the present situation, communication networks are not designed to the point that they can deal effectively with the intensity and the magnitude of operations that the applications require. As a result, this leads to performance degradation, such as crowding of the network channels with data bits, overheating of the components leading to circuit breakage, increasing latency, inefficient memory utilization and so on. Moreover, since mesh networks are the popular choice of topology on NoCs, such degradation could adversely affect the uniformity and the general structure of the mesh topology. NoCs do away with these problems with fast, scalable communication facilitated by small, distributed, packet-switched routers [6]. As shown in Fig. 1, NoC is a communication medium on an Integrated Circuit, linking the various IP cores on the chip. Its architecture consists of multiple segments of wires, routers, input/output ports, buffers, a crossbars and network interface, which are arranged in various topologies like the mesh and torus. A network interface (NI) module transforms data packets generated from the client logic (processor cores) into fixed-length flow-control digits (flits) [10].

In the design of NoCs, mainly the router architecture and its baseline algorithm decide their area and power budget. For area saving and efficient usage of router buffers, shared queue is used in Wormhole technique [7]. But the problem of deadlock is possible. In another work, virtual channel router architecture is proposed in which each port is linked to

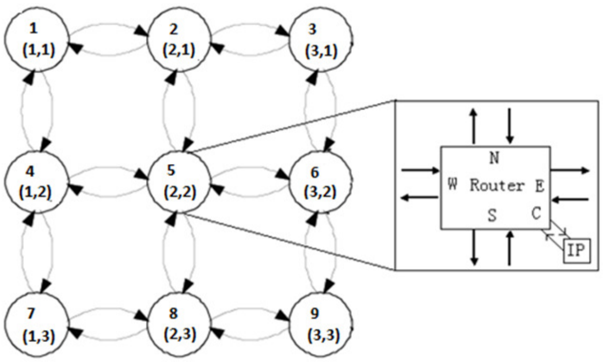


Fig. 1 A 2 dimensional 3x3 Network-on-Chip (NoC) with the mesh topology.

number of virtual buffers [11] – [12]. This may avoid deadlock but results with increased power and area. As a result, many buffers are active all the time, but only a few of them are utilized at a time, so that most of them are idle. In any case, NoCs forms a single and acutely fragile point of failure in a chip multiprocessor (CMP) system. Unlike the cores in a CMP, which are uniform, distributed, and therefore inherently redundant, there is only one communication medium in the chip, that is, one integrated NoC, which constitutes a pronounced weakness in the presence of faults [13]. Unreliable silicon substrates and their accompanying transistors that have been scaled down by a large factor, threaten the reliability and efficiency of NoC infrastructure, where a single transistor failure could be a severe liability unto the malfunctioning of the entire system [14]. The occurrence of failures in such technologies is expected to become frequent, given the range of their applications, leading to system malfunction and even causing security flaws [15].

Several research works have been done to develop efficient routing algorithms on the NoCs. A protocol that is commonly employed in network routing is the distance-vector protocol [16], which necessitates that a router update its neighbors with topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead. The term *distance vector* refers to the fact that the protocol manages *vectors* (arrays) of distances to other nodes in the network, which is crucial to the success of the protocol.

One of the algorithms meant for mesh and torus topologies is Vicis, as proposed by Andrew DeOrio et al. [13]. Vicis employs a reconfigurable router architecture and a routing algorithm. It takes advantage of the redundancy inherent in on-chip networks. As a distributed in-hardware solution, Vicis has the advantage of being able to tolerate many faults, including failures in the components. Vicis thus enables graceful performance degradation when transistors inevitably fail.

Next the XY routing algorithm, as proposed by Wang Zhang et al. [17], is a distributed deterministic routing algorithm. For a NoC with the two dimensional mesh topology, each router can be identified by its X and Y coordinates. The XY routing algorithm compares the current router address, say (Cx,Cy), to the destination router address, say, (Dx,Dy), of the packet. The destination address is mentioned in the header flit. Flits are routed to the core port of the router when the address of the current router matches the destination address. The routing request for this packet will remain active until a connection is established in some future execution of the procedure in this router.

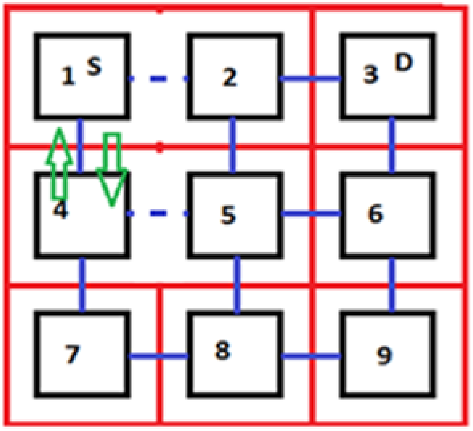


Fig. 2 A 3x3 mesh network with deadlock between routers 1 and 4.

A novel version of the XY routing algorithm, namely dynamic XY (DyXY) algorithm, which provides adaptive routing based on congestion conditions in the vicinity, and ensures both deadlock-free and livelock-free routing, was proposed by Ming Li et al. [18]. Using DyXY routing algorithm, the packet travels only the shortest path between the source and the destination. If multiple shortest paths are available then the router chooses the path for the packet depending on the congestion condition of every router in the network. The congestion condition of a router is determined by the number of cells that are occupied, in all input buffers, and each stress value is updated based on an event-driven mechanism.
Hence from the above studies, it is noticed that the Distance Vector algorithm does not determine the most convenient (the shortest) path between the source and the destination routers in a network when deadlocks (data moves to and fro between node 1 and 4 as the connection of the mentioned nodes with 2 and 5 are broken which is represented in Fig. 2) and livelocks are present, thus consuming precious time. The XY-Routing algorithm is also sensitive to deadlocks and gets the flow caught in the endless loop in the deadlock. Besides, Vicis, on the other hand, does not provide the shortest path, rather it provides a faultless path when failures occur in the network. While one algorithm is focused on determining the shortest path from the source node to the destination node, the other is focused on ensuring that the path of flow does not have any faults.

Given a situation of decreasing transistor reliability and increasing failures, the objective of the proposed work is to provide a solution that allows for smooth and uninterrupted data transfer between routers in the network even in the presence of faults such as wire and/or router damage, manufacturing defects and deadlocks. A routing algorithm with rerouting option is thus intended to be designed, which covers such faults - thereby raising fault tolerance. The next section explains the methodology of the proposed algorithm, what it is meant to do and about its various features.

## II. THE XY-CRUISE ALGORITHM

Inspired by Vicis' basic rerouting step and the DyXY routing algorithm, the proposed XY- Cruise algorithm uses the X and Y coordinates of the source and the destination routers, so as to obtain the shortest path between the two. When the said path is found to be faulty, data travels through the next nearest feasible path. This is the gist of the algorithm that combines finding the shortest path with rerouting mechanism. Deadlocks are avoided because once data has travelled a certain path that contains potential deadlocks, it doesn't retrace the same path. The term XY- Cruise was coined based on the routing function of the XY routing protocol coupled with the concepts of fast routing and rerouting in a faulty network. It is accompanied by code complexity that increases the processing speed. The data 'cruises' through successfully in the shortest time. Hence, the XY- Cruise algorithm came to be named as such.

The algorithm promises to:

1. Find the shortest path in a mesh topology based network
2. Reroute data within the network when other wires/nodes are broken
3. Compute the shortest path even in the presence of faults
4. Avoid infinite loops and deadlocks

The following  is the pseudocode of the proposed XY-Cruise algorithm.

```
sync_routers()
for every router
status=status_check(router,wires)
for every destination
basic_route_step(dest,status)

status_check(router, wires)
for this router
state=check_condition_wires(N,E,W,S)
return state

basic_route_step(dest,status)
current=source router;
routing_process:
while(dest is not routed)
if (dest is current)
routing_complete()
else
check_neighbors(router,status)
neighbor=find_nearest_neighbor(router,status)
data_flow(current,neighbor)
current=neighbor
jump to routing_process
```

The 'basic_route_step' function of the proposed XY-Cruise algorithm (inspired by Vicis' basic routing step function) is the crux of the entire XY- Cruise algorithm. It sets the source router as the current router, then begins the routing process ('routing_process' in the pseudocode). Here, as long as the destination router has not been reached (that is, the condition 'dest is current' is false), the statuses of the neighbours are checked and accordingly the one that lies on the shortest path is chosen so that data can flow from the current router (the 'source' in the beginning) to this neighbour. Then this 'neighbour' is set as the current router and the condition (dest is current) is checked and accordingly, the routing process continues or the routing is complete.

*Case 1 – Finding Shortest Path*

For a perfect 3x3 mesh network with no faults, the proposed XY- Cruise algorithm determines the shortest path for a particular source - destination node pair. The path travelled is displayed in Fig. 3. The functioning is based on the coordinates of the two routers involved. The code functions either with X priority or with Y priority. Here, Y priority is assumed. According to the algorithm:

1. Routers are numbered 1 to 9 like a computer's number-pad.
2. Data is to flow from router 1 (1,1) to router 9 (3,3)
3. Data flows from router 1 (1,1) to router 4 (1,2) because destination (3,3) has a higher Y co-ordinate
4. Then it flows from router 4 (1,2) to router 7 (1,3) because destination (3,3) has a higher Y co-ordinate
5. Then it flows from router 7 (1,3) to router 8 (2,3) because destination (3,3) has a higher X coordinate given Y coordinates are equal
6. Then it flows from router 8 (2,3) to router 9 (3,3) because destination (3,3) has a higher X coordinate given Y coordinates are equal

*Case 2 – Rerouting a Shortest Path for a Faulty Network (single fault)*

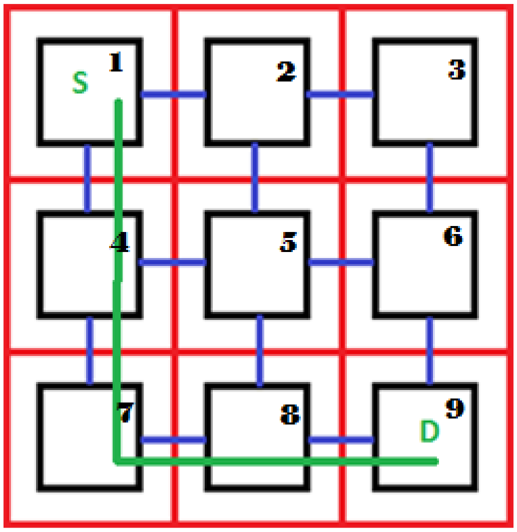


Fig. 3 A 3x3 Mesh Network with dataflow from (1,1) to (3,3) *(Case 1)*

For a 3x3 mesh network with a single fault, the XY-Cruise algorithm determines the next shortest path for a particular source - destination node pair when the fault occurs in the original shortest path. The path travelled is displayed in Fig. 4. Here, Y priority is assumed. Consider the dotted line (between routers 1 and 4) as the broken link in the network. There will be a slightly different path yet, it will still remain as the shortest path. According to the algorithm:

1. Data is to flow from router 1 (1,1) to router 9 (3,3)
2. Data flows from router 1 (1,1) to router 2 (2,1) because the link to router 4 (1,2) is broken (the Y-direction movement is not possible, so movement occurs in the X direction). It is rerouted here to the next shortest path
3. Then it flows from router 2 (2,1) to router 5 (2,2) because destination (3,3) has a higher Y co-ordinate (Y priority)
4. Then it flows from router 5 (2,2) to router 8 (2,3) because destination (3,3) has a higher Y coordinate
5. Then it flows from router 8 (2,3) to router 9 (3,3) because destination (3,3) has a higher X coordinate given Y coordinates are equal
6. In case X coordinate is given higher priority, the path travelled is:
   1 -> 2 -> 3 -> 6 -> 9

The path travelled is of the same length in either case as the number of routers involved in the routing process is five including source and destination.

*Case 3 – Avoiding Deadlocks*

For a faulty 3x3 mesh network with two faults (the links between the routers 1-2 and 4-5), the XY-Cruise algorithm determines the next shortest path for a particular source - destination node pair when the fault(s) occurs in the original shortest path and also when a possibility of looping (deadlock) occurs while trying to find the shorter path.

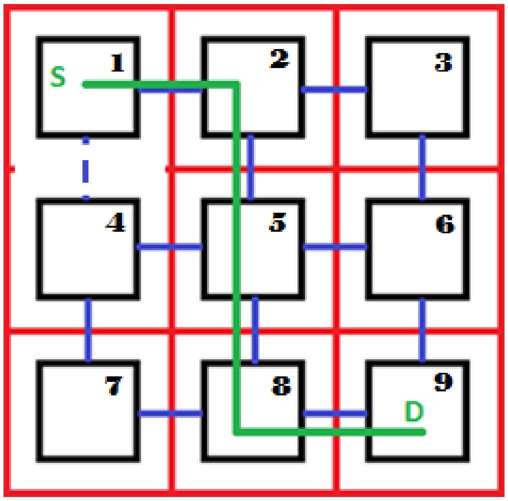
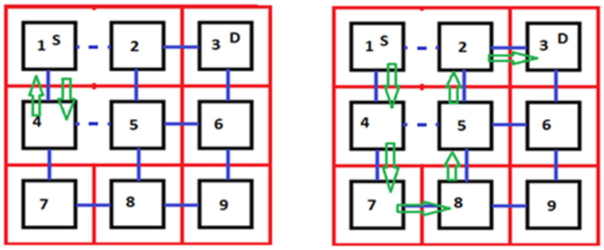


Fig. 4 A 3x3 Mesh Network with dataflow from (1,1) to (3,3) with a single fault *(Case 2)*



(a) Deadlock (Conventional XY-Routing algorithm)

(b) No deadlock (Proposed XY-Cruise algorithm)

Fig. 5 Faulty network with deadlock problem and its solution *(Case 3)*

For instance, consider the conventional XY-routing algorithm. When data travels from router 1 to router 4 and then when it wants to reach router 3 since the links to router 2 and router 5 are broken, the XY-routing algorithm tries to jump to the router closer to the destination, as shown in Fig. 5a. Thus, taking Y priority, data travels again to router 1 (1,1) because it is closer to router 3 (3,1) than router 7 (1,3) is. In the next step, since data can't travel to router 2 (2,1) because the link is broken. Thus it goes to router 4. An infinite loop arises as a result.

However, in the XY-Cruise algorithm, router 1 (1,1) is 'locked' – that is, data does not travel back to this router - once the code discovers that it is a potential deadlock-inducing router. So, it reroutes accordingly to router 7 (1,3) and then the shortest path is computed from there on. Thus, deadlocks and loops are avoided that way. The path travelled is displayed in Fig. 5b. Here, Y priority is assumed. Consider the dotted line as the broken link in the network. There will be a slightly different path yet, it will still remain as the shortest path. The Fig. 5a demonstrates a deadlock-case when the conventional XY-routing algorithm is employed, while the problem is resolved with the proposed XY-Cruise algorithm, as seen in Fig. 5b.

## III. RESULTS AND COMPARISON

In a GCC 64-bit C++ compiler, the XY-Cruise algorithm took lesser time to establish a shortest path flow of data in a 3x3 mesh for a sample case of flow between routers 1 and 9, as compared to the conventional Distance Vector algorithm (DVA), under the same conditions.

### *A. Distance Vector Algorithm (~90 ms)*

- 1 to 2 to 3 to 6 to 9 -> Weight: 4
- 1 to 4 to 7 to 8 to 9 -> Weight: 4
- 1 to 2 to 3 to 6 to 5 to 8 to 9 -> Weight : 6
- Time required to complete result: 2 x 4 cycles + 1x 6 cycles + other possible paths

### *B. Proposed XY-Cruise Algorithm (~20 ms)*

- 1 to 4 to 7 to 8 to 9: Weight 4 (or)
- 1 to 2 to 3 to 6 to 9: Weight 4
- Time required to compute result: 4 cycles

From the above observations, it can be inferred that the proposed XY-Cruise algorithm utilises lesser clock cycles to execute (lesser by at least 10 clock cycles) than the conventional Distance Vector algorithm.

Table I shows the results of compiling the XY-Cruise Algorithm on the GCC compiler for transfer of 4 bits of data, where up to four faults occur in the network. The distance vector algorithm consumes the same time no matter how it travels because of its multi-path nature. Be it faulty or faultless, all possible paths are calculated and the ideal path is selected based on least weight of data travel.

As can be observed, the XY-Cruise algorithm executes faster than the Distance Vector algorithm on a GCC compiler for routing data of 4 bits, by nearly 4.48 times for a faultless network. That is, computation time is reduced by 77.6%when the proposed XY-Cruise algorithm is employed. Similarly, computation time is reduced by 76.5%, 71.9%, 68.4% and 66.7% in the presence of a single, two, three and four faults respectively, in the network. Fig. 6 shows the results of compiling and executing the codes for the Distance Vector and XY-Cruise algorithm (with and without deadlocks) for routing data of sizes 4 and 8 bits.

TABLE I
Comparison of the time taken by the XY-Cruise and Distance Vector algorithms

| Number of faults | Time in milliseconds (ms) | |
|---|---|---|
| | Proposed XY-Cruise Algorithm | Distance Vector Algorithm |
| 0 | 20.7 | 92.8 |
| 1 | 21.8 | 92.8 |
| 2 | 26.0 | 92.8 |
| 3 | 29.3 | 92.8 |
| 4 | 30.9 | 92.8 |

TABLE II
Comparison of the time taken by the proposed XY-Cruise and Distance Vector algorithms

| Number of faults | Time taken in microseconds (us) | |
|---|---|---|
| | Proposed XY-Cruise Algorithm | Distance Vector Algorithm |
| 0 | 0.11 | 0.47 |
| 1 | 0.11 | 0.47 |
| 2 | 0.13 | 0.47 |
| 3 | 0.15 | 0.47 |
| 4 | 0.16 | 0.47 |

For Fig. 6 let the following denotes the legend:

Series 1: Distance Vector algorithm without deadlock
Series 2: XY-Cruise algorithm without deadlock
Series 3: XY-Cruise algorithm with deadlock

As the compilation results shown in Fig. 6, the proposed XY-Cruise algorithm is faster than the conventional Distance Vector Algorithm by slightly more than three times where the data to be routed contains 4 bits and approximately three times for the transfer of 8 bits of data. The slight variation arises due to the difference in bus sizes of the bit-streams and the involved complexity in routing them without errors or data loss. Table II shows the observations recorded when the XY-Cruise Algorithm (for routing data of 4 bits' size), was coded in Verilog, on the behavioural data flow model, and synthesized on Vivado 2014.4 and burnt into a Zynq 7000 Board.

**From the power report generated for XY - Cruise algorithm Total On-Chip Power (W) = 0.119*
**From the power report generated for distance vector algorithm Total On-Chip Power (W) = 0.124*

It has been observed that the power consumed by the Zynq 7000 board when programmed with the proposed XY-Cruise algorithm is 0.119W, as compared to the corresponding value of 0.124W in the case of the conventional Distance Vector algorithm. That is, a reduction in power consumption of 4% occurs when the XY-Cruise algorithm is employed.

Also in hardware level the following observations are made and consolidated as follows: The computation time falls by 76.5% when the XY- Cruise algorithm is employed for a faultless network, and by 76.5%, 72.3%, 68% and 65.9% for the occurrence of a single, two, three and four faults in the network, respectively.

Next when comparing the proposed XY- Cruise algorithm with Vicis in terms of overcoming deadlocks, slight improvement in the performance of the proposed work is observed and tabulated in Table III. This improvement is due to the fact that deadlocks which are not covered by the Vicis algorithm are resolved by the proposed algorithm.As Table III shows, on an average, the proposed XY- Cruise algorithm has 0.63% higher fault tolerance than Vicis.

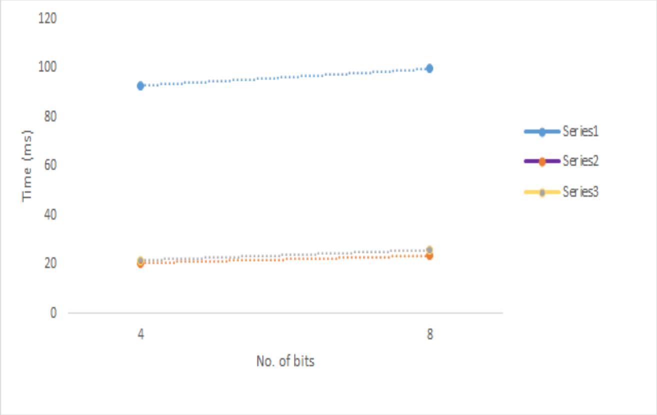


Fig. 6 Graph showing comparison of compilation times of Distance Vector and XY-Cruise algorithms

TABLE III
Comparison of the success rates of the XY-Cruise and the Vicis algorithms

| No. of Faulty Connecting Wires in Between | Percentage Success (XY-C) | Percentage Success (Vicis) |
|---|---|---|
| 0 | 100 | 100 |
| 1 | 99.91 | 99.65 |
| 2 | 99.83 | 99.31 |
| 3 | 99.56 | 98.87 |
| 4 | 96.85 | 95.93 |

*Percentage success= [1- (Number of unsuccessful cases/ Total number of cases)] *100*

For the situation where deadlocks occur, Fig. 7 shows the comparison of the time taken for execution of the Vicis and the proposed XY-Cruise algorithms for routing data of 4 and 8 bits.
For Fig. 7, let the following denotes the legend:

Series 1: Vicis algorithm with deadlock
Series 2: XY- Cruise algorithm without deadlock
Series 3: XY- Cruise algorithm with deadlock

We take the observations for the Vicis algorithm to be random (0 in the graph) because Vicis does not give any fixed value or even a range of values for routing data, considering that it was not meant to provide the shortest path in the event of deadlocks, rather only a faultless path.

## IV. CONCLUSION

The XY-Cruise algorithm has been presented as a solution to improve router performance in a network in terms of speed and fault tolerance, in the event that faults like wire/router damages and/or deadlocks occur, which is a frequent case due to nanotransistors. It combines the ideas of the basic routing function of the Vicis algorithm, which reroutes the network if and when faults occur, with the concept of XY routing (using the X and Y coordinates), specifically the Dynamic XY routing algorithm, in order to provide the shortest path from the source to the destination .As compilation results using a GCC compiler show, the proposed XY- Cruise algorithm is faster than the conventional

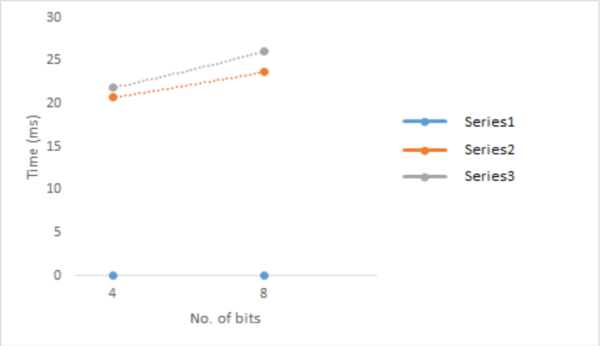


Fig. 7 Graph showing comparison of compilation times of XY-Cruise and Vicis algorithms

Distance Vector Algorithm by slightly more than three times where the data to be routed contains 4 bits and approximately three times for the transfer of 8 bits of data. When synthesized and burnt into a Zynq 7000 board for routing 4 bits of data, the results show that the XY- Cruise algorithm is faster than the Distance Vector Algorithm by approximately three times when no faults occur in the network, and by around 3, 2.5, 2.1 and 1.8 times for the occurrence of a single, two, three and four faults in the network, respectively.

With time, as the applications of network routing become more and more complex, solutions need to be found to allow for such functions to be performed while also handling the increasing occurrences of failures, as and when they do. The proposed XY-Cruise algorithm is a small step in this direction and hopefully, in the future, this algorithm can be improvised to work effectively and efficiently for router mesh topologies of orders greater than five, and also for other topologies. Also, following this train of thought, an XYZ-Cruise (for networks of three dimensions) could be developed in the future.

## REFERENCES

[1] M. A. Alam, “A critical examination of the mechanics of dynamic NBTI for PMOSFETs,” in *Proc. IEDM*, March 2003, pp. 32 – 35.

[2] Timothy G. Mattson, Rob F. Van der Wijngaart, Michael Riepen, Thomas Lehnig, Paul Brett, Werner Haas, Patrick Kennedy, Jason Howard, Sriram Vangal, Nitin Borkar, Greg Ruhl and Saurabh Dighe “The 48-core SCC processor: The programmer’s view,” in *Proc. of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2010, pp. 1- 11.

[3] Bell, S. et al, “TILE64 processor: A 64-core SoC with mesh interconnect,” in *Proc. ISSCC2008,* Feb. 2008.

[4] Vangal, S. R. et al., “An 80-tile sub-100-w teraFLOPS processor in 65-nm CMOS,” *IEEE Journal of Solid-State Circuits*, Vol. 43, No. 1, 2008, pp. 29 - 41.

[5] Multi-core Processor, https://en.wikipedia.org/wiki/Multi-core_processor (2017)

[6] William J. Dally and Brian Towles, “Route packets, not wires: On-chip interconnection networks,” in *Proc. DAC 2001*, pp. 684–689.

[7] Pierre Guerrier and Alain Greiner, “A generic architecture for on-chip packet-switched interconnections” in *Proc. DATE 2000*, pp. 250–256

[8] Yatin Hoskote , Sriram Vangal, Arvind Singh, Nitin Borkar and Shekhar Borkar, “A 5-GHz mesh interconnect for a teraflops processor,” *IEEE Micro*, 2007, Vol. 27, No. 5, pp. 51–61

[9] Shashi Kumar et.al.,“A network on chip architecture and design methodology,” in *Proc. of IEEE Computer Society Annual Symposium* , VLSI 2002, pp. 117–122.

[10] Networks on Chips: Structure and Design Methodologies, https://www.hindawi.com/journals/jece/2012/509465/ (2017)

[11]Mehdi Modarressi, Arash Tavakkol and Hamid Sarbazi-Azad, “Virtual point-to point connections for NoCs,” *IEEE Trans. Computer - Aided Design of Integrated Circuits and Systems*, Vol. 29, No. 6, 2010, pp. 855–868.

[12] N.V, Anjali and Somasundaram, K., “Design and Evaluation of Virtual Channel Router for Mesh-of-Grid based NoC,” in *International Conference on Electronics and Communication System, 2014,*

[13] Andrew DeOrio, David Fick, Valeria Bertacco, Dennis Sylvester, David Blaauw, Jin Hu, Student Member, and Gregory Chen, “A Reliable Routing Architecture and Algorithm for NoCs”, *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, Vol. 31, No. 5, May 2012, pp.726 – 739.

[14] Giovanni De Micheli, “Reliable communication in systems on chips,” in *Proc.41st DAC2004.*

[15] Andrea Pellegrini, Valeria Bertacco and Todd Austin, “Fault-based attack to RSA authentication,” in *Proc. DATE 2010.*

[16]Distance-vector routing protocol,https:/ /en.wikipedia.org/wiki/Distance-vector_routing_protocol (2016)

[17] Wang Zhang, Ligang Hou, Jinhui Wang, Shuqin Geng and Wuchen Wu, “Comparison Research between XY and Odd-Even Routing Algorithm of a 2-dimension 3x3 Mesh Topology Network-on-Chip”, *Global Congress on Intelligent Systems, 2009*, pp. 329 – 333.

[18] Ming Li, Qing-An Zeng and Wen-Ben Jone, “DyXY - A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip”, in *Proc. 43rd ACM/IEEE DAC,* 2006, pp. 849-852.