Source Hotspot Management in a Mesh Network on Chip

Sujay B Shaunak[1], Shashank S Rao[1], Ajay S[1], Satya Sai Krishna Mohan G[1], Krutthika H K[1], Ananda Y R[2], and John Jose[2]

[1] Dept. of ECE, Dayananda Sagar College of Engineering, Bengaluru, India
[2] MARS Lab, Dept. of CSE, Indian Institute of Techonology Guwahati, India
{sujayshaunak, shashankrao729, ajaysrinivasa96, sai.krishna521}@gmail.com
{krutthika.hk09, yrananda}@gmail.com, johnjose@iitg.ac.in

Abstract. Network-on-Chip helps to accomplish greater throughput in multi-core chips. In a multi-core chip, each core parallelly processes multiple applications thereby increasing the overall processing capability of the chip. One of the major concern in this field is managing congestion on the network. There are many reasons for congestion, one of them is hotspots, which has been considered in this paper. The applications on a multi-core architecture that operates on large amount of data and computation may create hotspots. These hotspots introduce congestion on the network and increase the latency of packets that pass through them. Our solution to hotspots, identify the source hotspots and decrease inflow of packets into the hotspots, thereby reducing the network pressure where hotspots are present. The congestion control scheme is a threshold based approach that dynamically evaluates the presence of hotspots on the network and a routing algorithm to effectively route the packets away from the hotspots. Our experimental results show that the packets are routed away from the source hotspots and the packet latency of the network is effectively reduced.

Keywords: Congestion Management · Odd-Even routing · Threshold

1 Introduction

The necessity for better performance is satisfied by increasing the number of processing cores on the chip. These multi-cores upon connecting on the conventional bus architecture lead to scalability issues. The time required for each core to be bus master and complete their transaction on the bus, would increase exponentially with increase in the number of cores. To tackle this issue and to simultaneously increase the number of cores on a chip, the concept of Network on Chip (NoC) was introduced [14]. There are many network topologies like mesh, torus, hypercube, octagon and fat-tree [15]. Using NoC architectures, the cores address cache misses and coherence transactions. Each router has a connection to one or many cores which routes the various packets of cache misses and coherency issues, concurrently. A router is a device that comprises of input/output

ports along with buffers to hold packets and the logic used to route the packets through the network. The packets that are generated follow a certain path to reach its destination core through the network of routers. The routing algorithm takes into consideration the information present in the packet and determines the next router a packet has to hop to depending on the routing algorithm. In classical XY routing algorithm, the packet first travels in the X (Horizontal) direction, then travels in the Y (Vertical) direction.

A major concern with any algorithm is that, as the traffic on the network increases the latency of the packets also increase. Latency is the time taken for the traversal of the packet from its initialization, till it's retirement. There is also a high possibility that the packet can be deadlocked: the packet cannot make for progress from a router because it is unable to obtain a buffer space in the downstream router. There is also a problem of livelock of the packets: the packet is continuously hoping from router to router but making no progress towards the destination. To overcome livelocks, the implementation of priority-based routing shows promising results. To avoid deadlocks, different routing algorithms with turn restrictions like North-last or East-first, odd-even are proposed.

In a real-time multi-core environment many applications can run simultaneously on different processing cores. Depending on the application, the number of cache misses generated by a single core may vary. When there are very few of those packets generated, most routing algorithm will be able to handle these packets without any deadlocks or livelocks. But, when there is an increase in the number of packets generated, traffic on the network increases which creates congestion. Congestion can be characterized by burstiness, injection distribution and hop distance. Burstiness is characterized by the rate of packet generation from a specific processing core (High, Moderate or Low). Injection distribution is characterized by the localization of traffic (Hot-Spot or Evened-Out). Hop distance is characterized by the proximity of the destination to the source (Local or Global) [1].

Our novel research proposal is trying to address source hotspot traffic. In source hotspot traffic, one or more routers produce a higher number of packets in comparison to others routers in the network. This results in higher traffic through a certain router. If the packets are routed through these routers, the congestion increases causing higher latency of the packets. Unless the only path for a packet is through the hotspot, the packet should be routed to any of the other possible paths to reach its destination. Thus, the latency of the deflected packets is improved. To validate our work, we are simulating our algorithm on cycle accurate simulator BookSim2.0 [16]. The benchmark suite we are using to support our test results is SPEC CPU 2006 benchmark. In addition, hardware implementation has been done for the proposed algorithm by using Zedboard hardware FPGA kit and Xilinx Vivado 2016.2.

The paper is organized as follows. Section II discusses the related works. Section III describes the motivation. Section IV discusses the proposed algorithm and implementation details. Section V gives the result and analysis. At the end, Section VI concludes the work.

2 Related Works

Link et al. [3] proposes to dynamically reconfigure the hotspot inducing computations to different cores periodically so as to even-out the thermal impact of the hotspots on the chip. Though the dynamic shifting of the hotspot does even-out the temperature, this technique cannot be applied to multithreaded applications as the resources needed for the applications cannot be split off to another core. Kakoulli et al. [9], proposed an Artificial Neural Network to predict where a hotspot might occur and using DOR-XY routing to resolve the hotspot. They claim that their algorithm works with an accuracy ranging from 65% to 92% with the overhead of the neural network not exceeding 5.06%. [8] provides an improvement to their previous paper [9] by reducing the network latency and enhancing the throughput by 81% for an overall hardware overhead of 11.4%.

Gupte et al. [10] proposed hot modules being swapped with cool modules to decrease the thermal effect on the chip. This method employs swapping of a hot module with any cool module available on the design under the consideration of only a single hotspot. [5] introduces Weighted Order Toggle which assigns weights to the links and sets the source-destination pairs as XY or YX routed to minimize network capacity for certain routing algorithms. They claim to balance the load on the links by repeated reconfiguration.

3 Motivation

Congestion that is encountered in single set turn restriction algorithms, such as north-last and east-first are non-existent in Odd-Even algorithm. Due to the above reason we choose Odd-Even algorithm. Considering a single set turn restriction algorithm like north-last, when traffic increases on the network the number of packets converging at a single router to travel north as their last turn increases. This in turn increases the congestion along that column which affects the latency of the packets traversing through or along that column. Just with turn restrictions that are present in Odd-Even routing algorithm, the problem of deadlock is avoided.

Alfaraj et al. [11] proposed HOPE (HOtspot PrEvention) algorithm which throttles packets at the source if the packet is destined to a destination hotspot. The hotspot is measured by checking if the destination router is receiving more flits than a certain threshold and then flags it as a hotspot. In [2] the hotspot mitigation proposed addresses destination hotspots. The technique used to calculate if a router is a hotspot or not is to use a single counter per port (four counters per router) and increment the counter if the packet is destined for that router or not. This counter value is compared to a fixed value and the router is flagged as a hotspot if the value is greater than the fixed value. The drawback in these techniques is, if the injection rate of the packets increases, then the number of packets on the network and the packets destined to a single router would significantly increase, thereby always flagging the router as a hotspot. This would cause a major problem on the network.

We have come up with a new approach to address the hotspot problem. This provides motivation to find a solution effectively to avoid hotspots in the path of the packet and decrease the latency thereby decreasing the network latency.

4 Proposed Work

The conventional method we consider is minimal Odd-Even routing algorithm [12]. In this algorithm the packet is always routed to take a minimal path from source to destination, adhering to the turn restrictions. This algorithm does not take into account the presence of source hotspots when routing a packet. This causes the packets to pass through a hotspot, thereby increasing the latency of those packets unnecessarily. This problem can be solved by identifying the source hotspots and rerouting the packets away from the hotspots. These approaches form the basis of our algorithm.

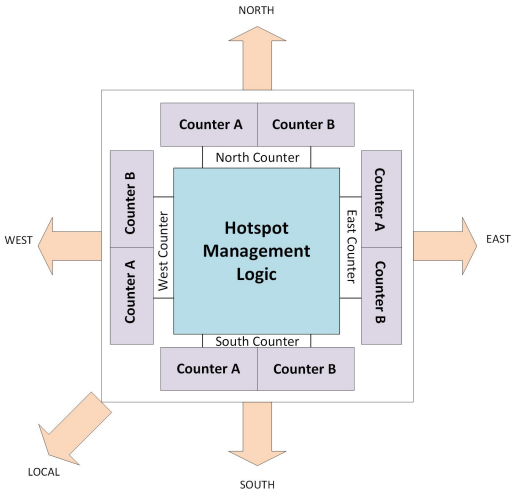
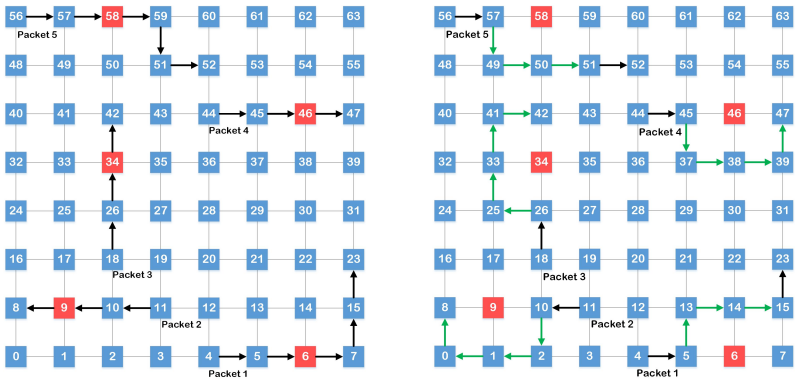


Fig. 1: Architectural changes proposed in the router.

Our simple approach proposes the usage of two counters for every port of each router as shown in Fig.1, one to count the number of incoming packets originating from the neighboring router (Counter A) and the other to count the number of incoming packets that do not originate from the neighboring router (Counter B). We use 2 conditions to flag a router as a hotspot. If both the conditions are satisfied only then the router is flagged as a hotspot. Firstly, we use a fixed value (hotspot threshold; HT) to check if Counter A is greater than HT. Secondly, we use a ratio of Counter A to Counter B which if exceeds a fixed ratio (ratio threshold; RT), flag that router as hotspot.

Algorithm 1 Algorithm for HAV (Hotspot AVoidance)

Input: Current core (cur) (C_x, C_y), Destination core (dest) (D_x, D_y), Source core (src), incoming packets from neighboring cores, CW = cycle window, HT = hotspot threshold, RT = ratio threshold, C_A [E/W/N/S] = Counter A, C_B [E/W/N/S] = Counter B and inport = input port packet came through
Output: Destination core router for all the incoming packets
if src is from one of the immediate neighbor routers **then**
Increment C_A for (east/west/north/south) the respective port
else
Increment C_B for (east/west/north/south) the respective port
end if
Calculate possible output ports from Algorithm 2
if (cur_cycle % CW == 0) **then**
if $C_A \geq$ HT AND $(C_A : C_B) \geq$ RT **then**
Flag that neighbour as hotspot
end if
end if
if Number of output ports in $Outputs > 1$ AND hotspot is present **then**
Mask the hotspot in $Outputs$
end if



(a) Conventional Algorithm. (b) Proposed Algorithm.

Fig. 2: Comparison of paths taken by proposed and conventional algorithm.

In reference to Fig. 2a and Fig. 2b, the following situations have been considered and the routes taken in the proposed and conventional algorithm have been explained.

1. **Packet 1 (Src: 4, Dest: 23):** Follows $4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 15 \rightarrow 23$ in the conventional method. As there is a hotspot in router 6 the proposed method avoids it by taking $4 \rightarrow 5 \rightarrow 13 \rightarrow 14 \rightarrow 15 \rightarrow 23$ which still has a minimal number of hops.

2. **Packet 2 (Src: 11, Dest: 8):** Follows $11 \rightarrow 10 \rightarrow 9 \rightarrow 8$ in the conventional method. As there is a hotspot in router 9 the proposed method avoids it by taking $11 \rightarrow 10 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow 8$ which takes 2 more hops than the minimal number of hops.
3. **Packet 3 (Src: 18, Dest: 42):** Follows $18 \rightarrow 26 \rightarrow 34 \rightarrow 42$ in the conventional method. As there is a hotspot in router 34 the proposed method avoids it by taking $18 \rightarrow 26 \rightarrow 25 \rightarrow 33 \rightarrow 41 \rightarrow 42$ which takes 2 more hops than the minimal number of hops.
4. **Packet 4 (Src: 44, Dest: 47):** Follows $44 \rightarrow 45 \rightarrow 46 \rightarrow 47$ in the conventional method. As there is a hotspot in router 46 the proposed method avoids it by taking $44 \rightarrow 45 \rightarrow 37 \rightarrow 38 \rightarrow 39 \rightarrow 47$ which takes 2 more hops than the minimal number of hops.
5. **Packet 5 (Src: 56, Dest: 52):** Follows $56 \rightarrow 57 \rightarrow 58 \rightarrow 59 \rightarrow 51 \rightarrow 52$ in the conventional method. As there is a hotspot in router 58 the proposed method avoids it by taking $56 \rightarrow 57 \rightarrow 49 \rightarrow 50 \rightarrow 51 \rightarrow 52$ which still has a minimal number of hops.

At low injection rates even if Counter A and Counter B is less in number and satisfies the ratio threshold (RT), the router is not flagged as hotspot due to not meeting with the hotspot threshold (HT) constraint, which means that the overall traffic on the network is not high enough for the router to become a hotspot. Whereas, at high injection rate, considering the hotspot threshold (HT) condition being satisfied, if the ratio of the counters is not met, which effectively tells us that the congestion on the network is evened-out, the router is not flagged as a hotspot. We observe the network for a fixed number of cycles which is defined as cycle window (CW) and at the end of every cycle window we evaluate the counters and flag routers as hotspots if necessary conditions are met. While checking for the hotspot, we normalize the counters so as to not lose the history of the packet count. The periodicity of this check and identification of hotspot depends on the number of cycles we monitor the network (cycle window; CW). The hotspot routers that were flagged remain in the same state till the next cycle window, when the conditions are checked again. Using this method, we can accurately identify source hotspots.

Utilizing Odd-Even turn restrictions we have modeled our algorithm to function minimally and actively avoid hotspots if present in the path of the packet. On encountering a hotspot, the algorithm makes the packet consider the non-minimal paths. Our routing algorithm calculates the possible output ports and assigns priority based on the minimality of the port with respect to the destination. By default the packet will take the minimal path following the Odd-Even turn restrictions. If the minimal path has a hotspot the packet takes an alternative path. In the case of only one available path, that single path is taken irrespective of a hotspot existing in that path or not. *Outputs* in Algorithm 2 is a set of the possible output ports a packet can take from a given router.

Algorithm 2 Algorithm for route calculation

if cur == dest **then**
  The current node is the destination node
else if $C_y == D_y$ **then**
  if not ($inport == west$ and $(C_x\%2) == 0$) **then**
    Add (north/south) appropriately to the set of *Outputs*
  end if
  Add west to the set of *Outputs*
else if $C_x == D_x$ **then**
  Add (east/west) appropriately to the set of *Outputs*
  if not (($inport == west$ AND $(C_x\%2) == 0$) OR ($D_y > C_y$ AND $(D_y\%2) == 0$) OR ($D_y < C_y$ AND $(C_y\%2) == 1$) **then**
    Add (north/south) appropriately to the set of *Outputs*
  end if
else if ($D_x > C_x$ AND $D_y > C_y$) **then**
  if not ((D_yC_y) == 1 AND $(C_y\%2) == 1$) **then**
    Add east to the set of *Outputs*
  end if
  if not ($inport == west$ and $(C_x\%2) == 0$) **then**
    Add north to the set of *Outputs*
  end if
  Add west to the set of *Outputs*
else if ($D_x > C_x$ AND $D_y < C_y$) **then**
  if not ($D_y < C_y$ AND $(C_y\%2) == 1$) **then**
    Add (north/south) appropriately to the set of *Outputs*
  end if
  Add west to the set of *Outputs*
else if ($D_x < C_x$ AND $D_y > C_y$) **then**
  if not ((D_yC_y) == 1 AND $(C_y\%2) == 1$) **then**
    Add east to the set of *Outputs*
  end if
  if not ($inport == west$ and $(C_x\%2) == 0$) **then**
    Add south to the set of *Outputs*
  end if
  Add west to the set of *Outputs*
else if ($D_x < C_x$ AND $D_y < C_y$) **then**
  if not ($D_y < C_y$ AND $(C_y\%2) == 1$) **then**
    Add north to the set of *Outputs*
  end if
  if not $(C_y\%2) == 1$) **then**
    Add south to the set of *Outputs*
  end if
  Add west to the set of *Outputs*
end if

Note: For a concise representation, the algorithm we have mentioned above does not include boundary conditions. The algorithm also does not include the logic for not sending a packet back through the same port it came in through. Both these conditions are considered and included in our simulation setup and results.

5 Experimental Work

5.1 Simulation setup and workload details

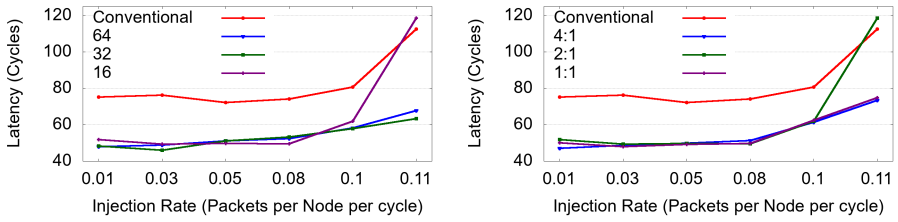
We implement our proposed algorithm on BookSim 2.0, a cycle accurate Network on Chip simulator. This software is versatile in configuring routing algorithm, router functionality, and flow control. We have run our simulations on an 8x8 mesh NoC with Hotspot injection. Hotspot injection is the type of synthetic injection of packets. This injection selects a set of routers randomly, which is flagged as hotspot. These routers inject a large number of packets for a set duration of time. This process is repeated to simulate sustained source hotspot formation. We use a uniform random traffic pattern to decide a destination for the packet which has been created by hotspot injection. The conventional method we have considered is Odd-Even minimal routing algorithm [12].

We compared our method with a set of multi-programmed workloads. We have used Gem5 simulator [18] to model the 64 core (8x8 mesh) CMP setup with CPU cores, cache hierarchy, and coherence protocols. Each core contains an out-of-order x86 processing unit with 64KB, dual ported, unified and private L1 cache. There is a shared L2 cache with a total size of 32MB. The block size of L1 and L2 are 32 bytes and 64 bytes, respectively.

We built 25 multi-programmed workloads, each with 64 applications selected from the SPEC CPU 2006 benchmark suite on gem5 simulator. We classified these workloads into 5 mixes (WL1 to WL5) based on the type of network injection intensity. The L1 cache misses that generate NoC packets are fed into BookSim2.0 simulator to simulate the operations on the network.

In our setup WL1 consists of applications (*bwaves* and *bzip*2) which have less number of hotspot events. Workloads WL2 and WL3 contains applications with low hotspot events (*bzip*2 and *gcc*) and applications with high number of hotspot events (*mcf* and *leslie*3*d*). Lastly, WL4 and WL5 comprises of applications with solely high hotspot events (*lbm*, *mcf*, *leslie*3*d* and *calculix*).

5.2 Analysis Under Synthetic Workloads



(a) Sensitivity analysis for Hotspot threshold (HT)

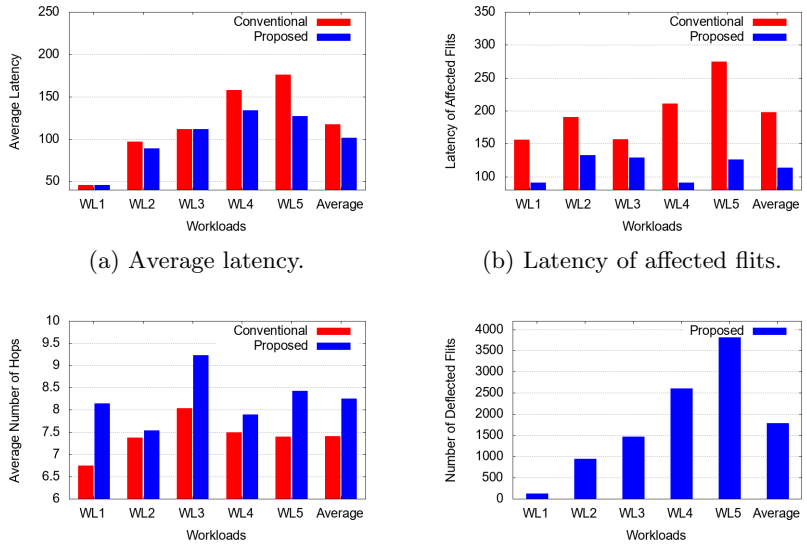
(b) Sensitivity analysis for Ratio threshold (RT)

Fig. 3: Sensitivity analysis

In Fig.3 we can see the comparison of our algorithm to the conventional algorithm in BookSim2.0. We consider the predefined hotspot threshold (HT) as 16, 32 and 64 while keeping the ratio threshold (RT) constant at 2:1 in Fig.3a. In Fig.3b we consider the ratio threshold (RT) as 1:1, 2:1 and 4:1 while maintaining the predefined hotspot threshold (HT) value at 32. In Fig.3a, the latency values for different hotspot thresholds are relatively similar. At higher injection rates, the latency for HT at 16 increases drastically due to false detection of hotspots. Similarly, the latency for HT at 64 increases by a small margin due to false hotspot identification. The threshold of 32 gives us the optimal result of lower latency. In Fig. 3b, the latency for RT at 2:1 is the highest while the RT at 1:1 and 4:1 give an optimal result.

In both the scenarios of changing the ratio threshold and hotspot threshold value, our algorithm provides better average latency in all the conditions. The difference between latencies when the ratios and thresholds change, is negligible. But, it can be observed that different ratios and different thresholds perform better at different injection rates.

5.3 Analysis Under Real Workloads



(a) Average latency.

(b) Latency of affected flits.

(c) Average number of hops.

(d) Number of deflected flits.

Fig. 4: Performance parameters for SPEC CPU 2006 benchmark.

Fig. 4 shows the simulation results of our algorithm on SPEC CPU 2006 benchmark using BookSim2.0. Fig. 4a shows the overall average packet latency. It can be seen that the average latency of our algorithm is lesser than the average latency of the conventional algorithm. Fig. 4b shows the latency of flits that pass through the hotspot in the conventional algorithm and latency of flits that are deflected in the proposed algorithm. Fig. 4c shows the average number of hops of the flits that are affected. Comparing Fig. 4b and Fig. 4c it is clear that while the latency of affected flits decreases, there is an increase in average number of hops. This is due to the fact that packets are taking a longer path to avoid the hotspots and since the hotspots are avoided the latency is decreased. Fig. 4d shows the number of flits that are deflected in the proposed algorithm.

In workload 1 (WL1), the average latency of the conventional method and proposed method is similar and also less, in comparison with the other workloads. The number of hotspots in this workload is also less along with the injection rate. The latency of affected flits is lower in the proposed algorithm as the hotspots are detected and avoided. For the low number of flits that are on the network at any point in time, the hotspots are correctly being identified and avoided.

In workload 2 (WL2), the number of hotspots increase and in turn the number of flits deflected increases. The same pattern in workload 1 (WL1) can be seen in the latency of deflected flits. The latency of affected flits in the proposed algorithm is lesser since it does not pass through hotspots whereas the average hops of affected flits increases and non-minimal paths are taken occasionally.

In workload 3 (WL3), the average latency between conventional and proposed method is almost equal. This is due to the drastic increase in the average number of hops. Due to this, the latency of the affected flits shows minimal change as compared to other workloads.

In workload 4 and 5 (WL4 and WL5), the difference in latency of affected flits between conventional and proposed algorithm is high since our algorithm gives the most optimal result. The injection rate in these workloads allows the algorithm to run at its best as the number of false detection of hotspot and undetected hotspots is the least compared to other workloads.

5.4 Hardware Analysis

With improvement in latency and throughput, the need to estimate the area and power overheads is essential. We calculate the area and power of our algorithm by modelling a router on Zedboard FPGA development kit [19] using Xilinx Vivado 2016.2 [20]. The data and control flows of the proposed router design is as follows. First, the packets received in the input buffers are checked.We then calculate the possible outputs ports using our routing algorithm with Odd-Even turn restrictions (Algorithm 2). We apply our hotspot aware counting algorithm (Algorithm 1). We assign input priority to all the packets from all the different directions by choosing one output port. Finally, we assign the output priority of the packets based on age. We implement the above logic in Verilog, synthesize and run the implementation of the design on Xilinx simulator. We then generate the bitstream and run our design on the Zedboard FPGA. We generate the area

and power reports of both our design and the conventional methodology. We compare the area and power overheads below.

Table 1: Hardware implementation details

	Look Up Tables(LUTs)	Flip-Flops (FF)	Power Consumed (mW)
Conventional Method	580	155	124
Proposed Method	722	195	126

We obtain a clock cycle time of 10ns with 50% duty cycle for our algorithm. We have compared the area and power values with the minimal Odd-Even router. Our algorithm has an overhead of 1.61% in power, 11.82% in Look-Up Tables and 2.96% in Flip Flops. This is because of the additional counters and combinational logic used for detection and avoidance of hotspot routers.

6 Conclusion and Future Works

Our paper proposed a method to manage the congestion created by the source hotspots. We explained the need for two check parameters for effective hotspot detection. We further discussed the problems that are encountered using a single check parameter and how our two check parameters of fixed hotspot threshold and fixed ratio threshold actively combats them. The proposed algorithm yielded better latency performance with mostly same number of hops but at the expense of a slight area and power increase. The benchmark simulation results showed that the latency of the packets that pass through the hotspot are improved, along with the overall average latency.

Our design can be extended to accommodate a variable threshold value and variable ratio threshold for the counters to more effectively and adaptively recognize hotspots. The results of the proposed algorithm implementation depend highly on the location of the hotspot. The conventional method works similar to ours when the location of hotspots is not critical but fails when the location of the hotspots is critical. Odd-Even routing algorithm has a few conditions where the turn restrictions prevent the packets from avoiding the hotspot. Improvements can be made in respect of changing the routing algorithm to a better deadlock free algorithm with turn restrictions that favours hotspot avoidance.

7 Acknowledgement

This work is supported in part by a grant from DST Government of India, SERB-ECR scheme (project number ECR/2016/212)

References

1. C. Wang, W. Hu and N. Bagherzadeh, "Scalable load balancing congestion-aware Network-on-Chip router architecture", *Journal of Computer and System Sciences*, vol. 79, no. 4, pp. 421-439, 2013.
2. R. S. Reshma Raj, A. Das and J. Jose, "Implementation and analysis of hotspot mitigation in mesh NoCs by cost-effective deflection routing technique," *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Abu Dhabi, 2017, pp. 1-6, 2017.
3. G. M. Link and N. Vijaykrishnan, Hotspot prevention through runtime reconfiguration in network-on-chip, *Design, Automation and Test in Europe (DATE)*, 2005, pp. 648-649.
4. W. Huang et al., HotSpot: a compact thermal modeling methodology for early-stage VLSI design, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501-513, 2006.
5. R. Gindin et al., NoC-Based FPGA: Architecture and routing, *International Symposium on Networks-on-Chip (NOCS)*, 2007, pp. 253-264.
6. L. Wang et al., A degree priority routing algorithm for irregular mesh topology nocs, *International Conference on Embedded Software and Systems (ICESS)*, 2008, pp. 293-297.
7. H. Wang et al., Thermal management via task scheduling for 3D noc based multiprocessor, *International SoC Design Conference (ISOCC)*, 2010, pp. 440-444.
8. E. Kakoulli et al., HPRA: A pro-active hotspot-preventive high-performance routing algorithm for networks-on-chips, *International Conference on Computer Design (ICCD)*, 2012, pp. 249-255.
9. E. Kakoulli et al., Intelligent hotspot prediction for network-on-chip based multicore systems, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 418-431, 2012.
10. A. Gupte and P. Jones, Hotspot mitigation using dynamic partial reconfiguration for improved performance, *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2009, pp. 89-94.
11. N. Alfaraj et al., HOPE: Hotspot congestion control for clos network on chip, *International Symposium on Networks-on-Chip (NOCS)*, 2011, pp. 17-24.
12. M. Tang, X. Lin and M. Palesi, "The Repetitive Turn Model for Adaptive Routing," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 138-146, 2017.
13. Wen-Chung Tsai, Kuo-Chih Chu, Yu-Hen Hu, Sao-Jie Chen, Non-minimal, turn-model based NoC routing, *Microprocessors and Microsystems*, Volume 37, Issue 8, Part B, 2013, Pages 899-914, ISSN 0141-9331.
14. L. Benini, G. De Micheli. "Networks on chips: a new SoC paradigm Computer", IEEE, Vol 35, pp. 70 - 78, Jan 2002.
15. Tatas, K., Siozios, K., Soudris, D. and Jantsch, A. (2014), Designing 2D and 3D Network on-Chip Architectures, 1st ed, New York: Springer-Verlag, 265.
16. N. Jiang et al., A detailed and flexible cycle-accurate network-on-chip simulator, International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013, pp. 86-96.
17. B. Nayak, J. Jose and M. Mutyam, "SLIDER: Smart Late Injection DEflection Router for mesh NoCs," 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, 2013, pp. 377-383.
18. Binkert, N., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M., Wood, D., Beckmann, B., Black, G., Reinhardt, S., Saidi, A., Basu, A., Hestness, J.,

Hower, D. and Krishna, T. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2), p.1.
19. Zedboard.org Zedboard. Available at: http://www.zedboard.org/product/zedboard
20. Xilinx.com Xilinx. Available at: http://www.xilinx.com/products/design-tools/vivado