Implementation and Analysis of Hotspot Mitigation in Mesh NoCs by Cost-Effective Deflection Routing Technique

Reshma Raj R. S.*, Abhijit Das† and John Jose†
* Dept. of IT, Government Engineering College Bartonhill, Trivandrum, India
† MARS Research Lab, Dept. of CSE, Indian Institute of Technology Guwahati, India
Email: reshmaraj26@gmail.com, {abhijit.das, johnjose}@iitg.ernet.in

***Abstract*—Network-on-Chip (NoC) serves as an efficient communication framework among the components of Chip Multi-Processors (CMPs). With increasing number of computation intensive applications communication between cores also increases, which creates high congestion resulting in network performance degradation. Handling congestion is a key network management issue in NoC. Hotspots are non-uniform traffic formation near cores where some cores need to handle a relatively higher traffic compared to others. Prolonged presence of these hotspots increases the communication latency of packets flowing through them. This work proposes a novel approach to identify destination hotspots and upon identification, packets are de-routed away from these hotspot cores using a cost-effective deflection routing technique. Experimental results show that in highly congested networks, our approach detects destination hotspots with great accuracy. De-routing of packets away from hotspots help them achieve congestion relief thereby decreasing the average latency of packets flowing in the network.**

***Index Terms*—Multicore Systems, Hotspot Traffic, Congestion Management, Flit Flow Analysis, Flooding.**

I. Introduction

Proving the correctness of Moores law in processor scaling till date, researchers continued proposing scalable architecture designs containing multiple processing cores on a single chip [1]. This increase in the number of cores demand an interconnection framework with efficient routing and flow control mechanisms. With these mechanisms in place, the processing cores are expected to experience minimum-latency and maximum-throughput for inter-core communications.

Network-on-Chip (NoC) is a widely adopted interconnection framework for satisfying the communication requirements of such multi-core systems, also called as Chip Multi-Processors (CMPs). 2D mesh [2] is the most preferred topology in NoC framework for its simple layout and short inter-core links.

In a generic CMP each core houses an out-of-order superscalar processor, a private L1 cache and a shared distributed L2 cache [3]. Each of these processing cores are connected to a router and all routers are interconnected with bi-directional links. Typically, L1 cache misses trigger packet generation. Packet exchange is the mode of communication in these NoC based CMPs where packet header contains control information and packet payload contains required data. Generally, a source core creates a packet with all required control information and injects it to the local router. Buffers in routers and handshaking signals between routers facilitate flow control and smooth packet movement between source and destination cores. Most NoCs employ wormhole switching [4], where each of these packets are divided into smaller flow control units called flits. The effort in breaking packets into flits is for saving router buffer size. On its way to the destination core, a packet might have to be forwarded through multiple intermediate routers where the employed routing algorithm decides on the outgoing link for every incoming packet in a given router.

Heterogeneous applications running in different cores of a CMP inject unpredictable traffic. With limited network resources like bandwidth, buffers, channels etc., often some NoC routers are flooded with packets [5]. These routers receiving more packets than they can actually handle results in hotspot formation. Hotspots are essentially cores which continuously send / receive packets to / from other cores leading to large delay in processing these packets. Additional factors for hotspot formation includes, dynamic and random resource demands, improper load balancing, sub-optimal application mapping, application unaware routing policies and random application migration.

Wormhole switching worsens the adverse effect of hotspots when pipelined flits quickly fill up upstream input buffers while waiting to get routed. The hotspots are then spatially spread across multiple routers blocking packets even at non-hotspot cores. Prolonged presence of these hotspots can make the NoC inoperable thereby taking the communication latency to a very high value. With increasing cores in modern CMPs, hosting diverse application across various cores, hotspot formation is both inevitable and unpredictable. To tackle NoC level congestion, different congestion-management techniques under adaptive routing strategies are proposed in the past. These generic proposals range from application scheduling [6], load balancing [7], to additional buffering [8] etc. Specific proposals on hotspot prevention and mitigation are also explored from [9] through [20].

978-1-5386-2880-5/17/$31.00 ©2017 IEEE

Contributing to the existing body of knowledge, this paper attempts to propose a novel deflection routing technique to mitigate the effects of destination hotspots. Destination hotspots are cores which continuously receive packets from a wide range of source cores. This technique is proposed for mesh based NoCs keeping in mind its enormous popularity. Our contribution in this paper can be summarized as follows:

- At regular intervals, we dynamically analyze the entire network traffic to identify possible destination hotspots that degrade network performance.
- We then apply a novel congestion management scheme with minimal deflection routing to de-route packets that had to pass through identified destination hotspots.
- We perform evaluation studies, compare our technique with conventional method and draw meaningful insights.

II. Related works

Till early 2000s; before NoC became a prevalent technology, it was all about generic congestion and thermal management techniques on chip. One of the earliest work specifically on hotspot prevention is by Link et al. [9], where it migrates hotspot inducing applications across the chip by periodically modifying their configurations. Though these dynamic shifting balances the overall chip temperature, it is not possible to shift hotspot in multithreaded applications. Since then, hotspots are an active area of computer architecture research.

Huang et al. [10] proposed an abstract self-heating power and temperature NoC model that can be used for analysis of thermal impacts during early design stages when layout and routing details are unavailable. Weighted Ordered Toggle [11] claims that it can be reconfigured over and again to balance link loads based on traffic patterns known apriori. A degree priority routing algorithm to minimize hardware cost in irregular mesh topology is also presented [12]. Here, routing path is dynamically chosen depending on the communication status of the next hop. For hotspot mitigation, dynamic reconfiguration capabilities of FPGAs can also be used [13]. When temperature of a given region exceeds a threshold, dynamic reconfiguration is used to swap the module thus evenly distributing the heat over FPGA surface.

A scheduling algorithm [14] for optimizing system performance under required temperature constraints is implemented with OS-level thread scheduling. HOPE [15] was proposed with the objective of designing an effective scheme for hotspot congestion detection at the earliest possible stage and regulate traffic destined to hotspot destinations thus avoiding saturation-tree formation in clos NoC. HPRA [7] [16], a Hotspot-Preventive Routing Algorithm pro-actively prevents the future formation of NoC hotspots by balancing the traffic using principles of artificial intelligence.

A technique that reduces peak temperature and the impact of hotspot in test applications by avoiding delivery of packets to NoC hotspot in each unicast step is explored [17]. Temperature aware re-routing strategy can reduce the impact of hotspots without compromising the performance benefits of mSWNoC [18]. TAPP [19] discuss an efficient application mapping by hierarchical bi-partitioning of cores to reduce hotspots. MinHotspot [20] focuses on explicit hotspot minimization in NoC. They have considered both computation and communication workloads in identifying hotspots and found feasible solutions for large-scale problems.

III. Motivation

The objective of handling hotspots can be either for thermal and power management or for congestion and communication management. Most of the available proposals handled hotspots for thermal and power [9] [10] [13] [14] [17] [18] [19] [20], while only a few considered the case of congestion and communication management [11] [12] [15] [16], despite the fact that overall NoC performance is more susceptible to network congestion than chip heating. Our work is focused towards congestion management.

Network hotspots can be tackled either with prevention or with mitigation. Among the existing techniques for congestion management [11] [12] [15] [16] (as identified earlier), all except [12] are towards prevention. Prevention sounds better but knowing hotspots apriori requires understanding the traffic patterns of heterogeneous applications, and that is an NP problem. For identifying and isolating cores that may lead to hotspot formation, the circuit complexity is very large. Moreover, incorrect hotspot identification results in underutilization of shared resources. Considering all these factors this work focuses on hotspot mitigation by de-routing the packets away from hotspot cores thereby providing congestion relief.

Hence, we are motivated to propose a novel congestion management technique with minimal deflection routing to mitigate the effects of identified destination hotspots. Analysis and evaluation statistics show that our technique offers significant reduction in average packet latency.

IV. The Proposed Work

Hotspot traffic leads to flooding of packets around the hotspot core resulting in high latencies for packets passing through it and its neighbours. For example, if more number of cores in a network want to access L2 cache sets mapped to a particular cache core then the demand of that core increases, which in effect becomes a destination hotspot. If a core is identified as a hotspot it is advisable to make the core free from those packets which are not destined to it.

In XY routing (conventional method) a packet will always follow a minimal path from its source to destination. Even if a core in the path is a hotspot, the packet has no other choice, it will have to go through the hotspot core by experiencing the additional delay. This will result in an increased overall latency of such packets. For controlling congestion at the hotspot, the packets that had to pass through that core need to be de-routed through a non-minimal path towards their destination. Hence, we consider the technique of deflection routing. Two basic steps involved in our approach.

1. Finding destination hotspots in the network.
2. De-routing packets away from those hotspots.

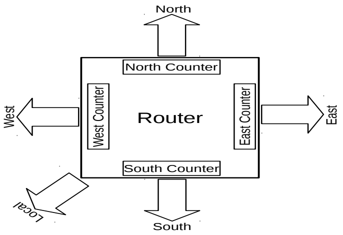

Fig. 1. Four hotspot detection counters kept in a router.

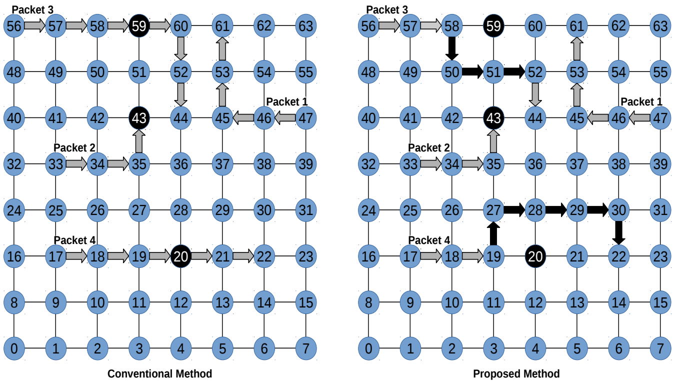

Fig. 2. Comparison of packet paths in conventional XY routing and our approach of hotspot aware deflection routing.

For a 2D mesh NoC with an 8×8 organisation we add four 9-bit counters per router to identify the hotspot cores. Each counter is meant for recording the traffic flow to its East, West, North and South neighbours as shown in Fig. 1. Appropriate counter is incremented if the packet received by a router is meant for the respective nearest one hop neighbour.

Hotspot location can vary over time depending on dynamic traffic footprint in the NoC, i.e. a core identified as hotspot in a given time frame need not be a hotspot in another time frame. So, we check for hotspots at regular intervals. At the end of each such *Compute Interval (I)*, the counters are decremented by right-shifting them to 2-bits (divided by 4) in order to avoid counter overflow and to preserve flit flow history.

The counter values associated with each router are checked at the end of each *Compute Interval (I)*. If the counter value is greater than a fixed *Hotspot Threshold (T)*, then the respective destination is identified as a hotspot by the adjacent router. After that for another one *Compute Interval (I)*, it will not forward any packet to that neighbour other than packets destined to that core. Instead it routes packets away from the hotspot core by a customized deflection routing technique. However, packets destined to hotspots will be forwarded to them. Here we alter the path of packets that are supposed to pass through hotspot core as per normal XY routing.

Consider Fig. 2, where the dark cores (20, 43, 59) are identified as hotspots by their respective adjacent cores. All other cores are non-hotspot cores. The arrows represent packet movement path from source to destination, and the dark arrows are the deflected path (taken in proposed method).

Consider the four packets moving around the network.

Algorithm 1: Proposed method of Deflection Routing

```
Consider a $K \times K$ network with $N = K^2$ cores
Input  : Current core$(C_x, C_y)$, Destination core$(D_x, D_y)$
Output: Selected output port
begin
    if $C_y != D_y$ then
        if $C_x < D_x$ and next core is hotspot then
            Select North
        end
        else if $C_x > D_x$ and next core is hotspot then
            Select South
        end
        else if $C_x == D_x$ and next core is hotspot then
            Select North or South
        end
        else if next core == previous core then
            Select North or South based on current core
        end
        else
            Select East or West based on current core
        end
    else
        if $C_x != D_x$ then
            if next core is hotspot then
                Select East or West
            else
                Select North or South based on
                current core
            end
        else
            Select Local
        end
    end
end
```

1) **Packet 1 (Src: 47, Dest: 61):** Follows the minimal path 47 → 46 → 45 → 53 → 61 in both conventional and proposed method as there are no hotspots in this path.
2) **Packet 2 (Src: 33, Dest: 43):** Follows the minimal path 33 → 34 → 35 → 43 in both conventional and proposed method even though 43 is a hotspot. This is because a packet is not deflected if its destination itself is a hotspot. This packet may experience delay in reaching 43.
3) **Packet 3 (Src: 56, Dest: 44):** In conventional method it follows the minimal path 56 → 57 → 58 → 59 → 60 → 52 → 44 even though 59 is a hotspot. But in the proposed method after core 59 is identified as a hotspot by core 58, packet 3 follows the de-routed minimal path 56 → 57 → 58 → 50 → 51 → 52 → 44.
4) **Packet 4 (Src: 17, Dest: 22):** In conventional method it follows the minimal path 17 → 18 → 19 → 20 → 21 → 22 even though 20 is a hotspot. But in the proposed method after core 20 is identified as a hotspot by core 19, packet 4 follows the de-routed minimal path 17 → 18 → 19 → 27 → 28 → 29 → 30 → 22.

Percentage Miss Rate	Benchmarks
Low MPKI (less than 5)	calculix, gobmk, gromacs, h264ref
Medium MPKI (between 5 and 25)	bwaves, bzip2, gamess, gcc
High MPKI (greater than 25)	hmmer, lbm, mcf, leslie3d

TABLE I
CLASSIFICATION OF BENCHMARKS BASED ON MPKIS

Workload #	SPEC CPU 2006 Benchmarks			
WL1	calculix(16)	gobmk(16)	gromacs(16)	h264ref(16)
WL2	calculix(16)	gobmk(16)	gamess(16)	gcc(16)
WL3	bwaves(16)	bzip2(16)	gamess(16)	gcc(16)
WL4	bwaves(16)	bzip2(16)	hmmer(16)	lbm(16)
WL5	hmmer(16)	lbm(16)	mcf(16)	leslie3d(16)

TABLE II
VARIOUS WORKLOAD MIXES

Since the deflected packets are taking additional hops to reach their destination, we enforce a priority mechanism such that they are no more penalised. In our approach the deflected packets are prioritized. A deflected packet is marked with an additional control flag in its header so that, in its de-routed path all routers will know that this is a packet deflected away from hotspot. We also maintain an extra field in the packet to store core number from where that packet is deflected. This is to avoid the re-routing of that packet again to the hotspot by other routers for whom that core is not a hotspot. Since the flow of packets to an identified hotspot is cut down from one of its neighbour, over next few cycles the hotspot core will experience a congestion relief. Our approach, by packet flow reduction to hotspots, allows the system to gain a relief from congestion. This modified congestion management technique results in reduced average packet latency, which improves the performance of the NoC system. Even though the deflected packets are travelling more hops, average flit latency is decreased. Also, it reduces the traffic flow to the hotspot core and helps the hotspot core to recover from hotspot scenario.

Algorithm 1 describes the deflection logic for packets that have to pass through the hotspot cores. In every cycle, packets traversing through routers are checked if they are approaching a destination hotspot. If the next router to be selected as per XY routing is a hotspot and is not a packet's destination, then the packets choose an alternate path. The algorithm is designed in such a way that it cross over the hotspot and does not come back; neither to the de-routed core nor to the hotspot core. This eliminates the possibility of deadlock.

V. EXPERIMENTAL STUDY

A. Simulation Setup

We use a cycle-accurate simulator, BookSim 2.0 [21] for the NoC simulation. It offers flexibility with a large set of configurable network parameters such as topology, routing algorithm and flow control.

To evaluate our approach multicore workloads of SPEC CPU 2006 benchmarks are used, which are classified according to their Misses Per Kilo Instructions (MPKI) on a 64KB L1 cache as shown in Table I. This is to classify the applications to different network injection intensity groups. The applications *calculix*, *gobmk*, *gromacs*, and *h264ref* have less than 5 misses per 1000 instructions, hence we group them under Low MPKI. Similarly, *bwaves*, *bzip2*, *gamess*, and *gcc* are in the Medium MPKI (between 5 and 25) group, and *hmmer*, *lbm*, *mcf*, and *leslie3d* are in High MPKI (greater than 25) group.

Based on this network injection intensity, we create 5 workloads consisting of SPEC CPU 2006 benchmark mixes as shown in Table II. Consider workload 1 (WL_1); where out of 64 cores that we model for the simulation, 16 cores run *calculix* benchmark, 16 cores run *gobmk*, 16 core run *gromacs* and last 16 cores run *h264ref*. Various combinations (application to core mapping) is done and the average of all these spatial mappings is plotted in all our results represented by workload i (WL_i). Similarly, other workload compositions (WL_2 – WL_5) can also be described.

We run 64 application instances of the respective workloads in gem5 architectural simulator [22] and extract the network events (L1 cache misses). The packets hence generated are fed to BookSim simulator to model the NoC traffic. Network statistics are collected and analysed. In this approach we consider an 8×8 2D-mesh NoC with 8 VC's per router input port. We use a 64-bit wide flit channel. Each cache miss creates 1 flit request packet and follow 4 flits reply packets.

B. Analysis on Hotspot Related Network Parameters

Fig. 3 shows the number of destination hotspots detected in each workloads using our approach. Among the workloads used, WL_1 generates few cache misses due to usage of Low MPKI applications in all of its 64 cores, hence few NoC packets are created. This amount for low congestion in the network, hence the number of destination hotspots are low. But WL_5 generates more number of misses and more NoC packets are created leading to high congestion. Hence as expected our algorithm identifies hotspots with great accuracy level.

Fig. 4 shows the overall latency of packets in the network. The concentration of destination hotspots increases with increase in congestion (like WL_5) which causes the network to be in high latency. Using our method after identifying the destination hotspots, the flits passing through those hotspots are de-routed such that it reduces the congestion at hotspot cores thereby reducing the latency of packets in the entire network. Results show that there is 5.24% average reduction in latency due to the proposed de-routing of flits.

Fig. 5 shows the number of flits that gets deflected due to destination hotspot cores in various workloads. Few number of flits gets deflected due to hotspot in case of low MPKI workloads (like WL_1 and WL_2), and as the congestion increases the number of flits deflected is high due to large number of hotspot cores.

Fig. 6 shows a comparative study of latency of flits that are passing through the destination hotspots in conventional method and the latency of flits that are deflected away in proposed method. This study reveals what is the change in

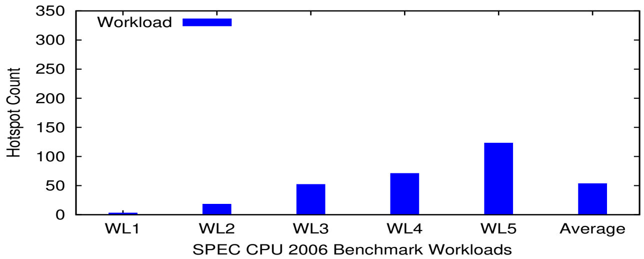

Fig. 3. Overall hotspot count

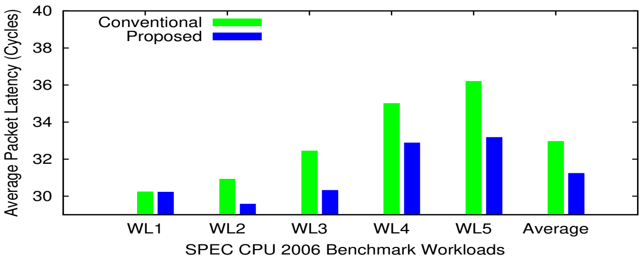

Fig. 4. Overall packet latency

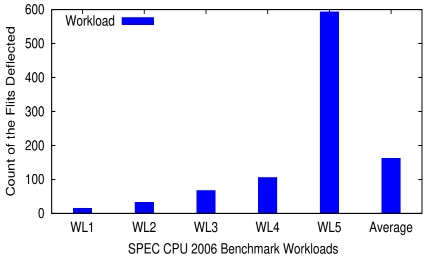

Fig. 5. Count of deflected flits

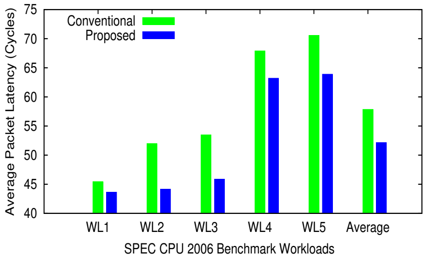

Fig. 6. Latency of flits passed through hotspot

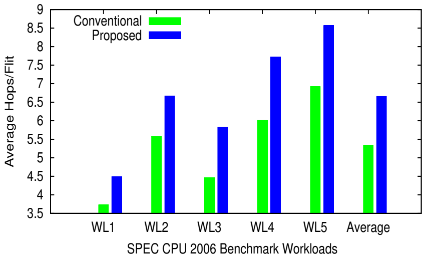

Fig. 7. Hops traversed by deflected flits

latency of flits that pass through hotspots in conventional method (there is no deflection here, all flits has to pass through the hotspots) and what will happen to the same flits if they get deflected in our proposed approach. From the graph it is evident that the latency of deflected flits gets decreased due to the choosing of a non-congested path. The graph also shows an average reduction of 9.86% in latency of such flits.

Fig. 7 compares the average number of hops/flit in forwarding through destination hotspot cores in conventional method with the average number of hops/flit that are deflected due to destination hotspot in proposed method. Since our approach is using the technique of deflection routing the de-routed packets have to travel few extra hops than in XY routing. From the above results it is evident that using our approach even if the deflected flits have to travel more hops (Fig. 7), the overall average flit latency is lower (Fig. 4).

C. Sensitivity Analysis of Design Parameters

For all the results discussed so far we have considered *Compute Interval (I)* and *Hotspot Threshold (T)* as 1024 and 256 respectively. Now to decide on the optimal *Compute Interval (I)* and *Hotspot Threshold (T)* values, sensitivity analysis is carried out by taking different values for these parameters. The *Compute Interval (I)* is fixed by studying how frequently the behaviour of hotspots changes, such that lower the interval, the computations are to be done continuously in short span of time which increases system overhead and higher the interval, it decreases system performance. Based on the performed simulation and analysis, we have identified an optimal value of 1024 clock cycles for *Compute Interval (I)*. Now keeping the *Compute Interval (I)* fixed to 1024, *Hotspot Threshold (T)* is varied and the differences in hotspot count and percentage reduction in average packet latency are recorded.

Fig. 8 shows the number of destination hotspots detected with *Hotspot Threshold (T)=128*. After every *Compute Interval (I)=1024*, the counter value of each core is compared and if it is found greater than 128 then that core is identified as a destination hotspot. Fig. 11 shows the comparison of overall packet latency obtained with *Hotspot Threshold (T)=128*. Likewise, Fig. 9 and Fig. 12 respectively shows the number of destination hotspots detected and overall packet latency with *Hotspot Threshold (T)=256* compared to conventional method. Similarly, Fig. 10 and Fig. 13 shows respective statistics with *Hotspot Threshold (T)=512*.

The above study clearly shows that with *Hotspot Threshold (T)=128*, the number of hotspots identified is much more. That means in this case cores with counter value greater than 128 are termed as hotspot core, which will be high in number. Since the number of hotspots are more, packets deflecting away from these hotspot cores are also more resulting in decreased system performance (Fig. 11). For *Hotspot Threshold (T)=512*, the number of cores that are crossing the threshold value is much lower thus there is no deflection at all. This makes the proposed system performance remain same as that of the conventional system (Fig. 13). However, it can be seen from Fig. 12 that setting *Hotspot Threshold (T)=256*, the proposed system significantly improves average packet latency. Thus comparing all these cases, the optimal value for *Hotspot Threshold (T)* is fixed as 256. However, depending on the performance requirements other values of *Hotspot Threshold (T)* can also be adopted.

VI. Conclusion

Hotspot formation leading to congestion in NoC gradually decreases system performance. Thus, hotspot detection and prevention has great importance in congestion management of

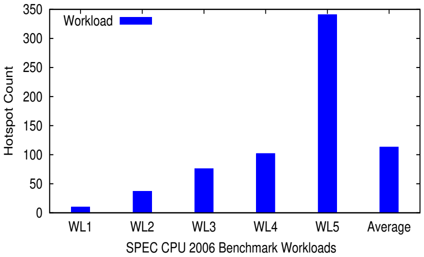

Fig. 8. Hotspot count (*I=1024, T=128*)

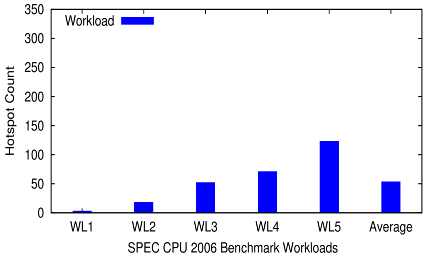

Fig. 9. Hotspot count (*I=1024, T=256*)

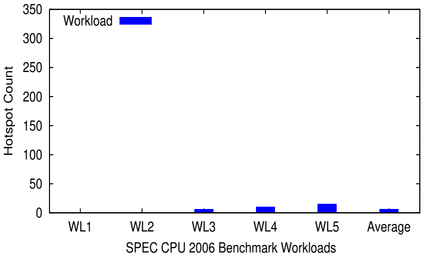

Fig. 10. Hotspot count (*I=1024, T=512*)

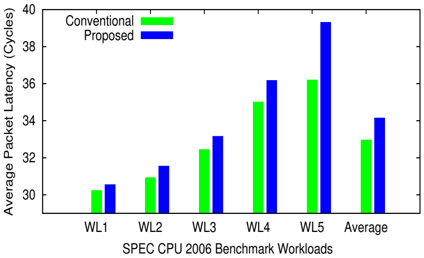

Fig. 11. Packet Latency (*I=1024, T=128*)

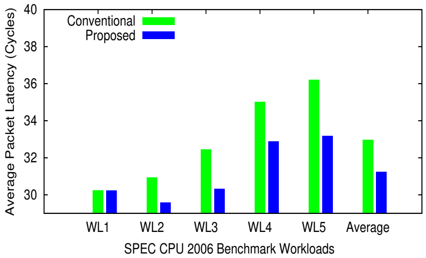

Fig. 12. Packet Latency (*I=1024, T=256*)

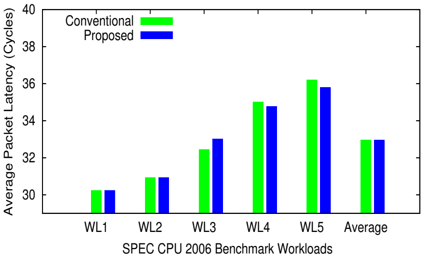

Fig. 13. Packet Latency (*I=1024, T=512*)

NoC. In this paper, we first introduced counters at each router for identifying the formation of destination hotspots. If a router identifies its neighbour as a hotspot, the packets that passes through that core are now de-routed using a deflection routing. Results showed great accuracy in hotspot identification and even though de-routing causes the packets to travel more hops than usual, the flit latency of those packets tends to decrease, which increases the overall system performance.

Future work includes identification of source hotspots which send more packets than others in the network and selection of dynamic hotspot threshold based on network behaviour for further improving the accuracy of hotspot detection.

References

[1] B. A. Nayfeh and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, 1997.

[2] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Annual International Conference on Supercomputing (ICS)*, 2006, pp. 187–198.

[3] É. Cota *et al.*, "NoC basics," in *Reliability, Availability and Serviceability of Networks-on-Chip*. Springer, 2012, pp. 11–24.

[4] W. J. Dally and B. P. Towels, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[5] E. Nilsson *et al.*, "Load distribution with the proximity congestion awareness in a network on chip," in *Design, Automation and Test in Europe (DATE)*, 2003, pp. 1126–1127.

[6] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, pp. 1–51, 2006.

[7] 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.

[8] U. Y. Orgas *et al.*, "Analysis and optimization of prediction-based flow control in networks-on-chip," in *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*. Springer, 2013, pp. 105–133.

[9] G. M. Link and N. Vijaykrishnan, "Hotspot prevention through runtime reconfiguration in network-on-chip," in *Design, Automation and Test in Europe (DATE)*, 2005, pp. 648–649.

[10] 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.

[11] R. Gindin *et al.*, "NoC-Based FPGA: Architecture and routing," in *International Symposium on Networks-on-Chip (NOCS)*, 2007, pp. 253–264.

[12] L. Wang *et al.*, "A degree priority routing algorithm for irregular mesh topology nocs," in *International Conference on Embedded Software and Systems (ICESS)*, 2008, pp. 293–297.

[13] A. Gupte and P. Jones, "Hotspot mitigation using dynamic partial reconfiguration for improved performance," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2009, pp. 89–94.

[14] H. Wang *et al.*, "Thermal management via task scheduling for 3D noc based multi-processor," in *International SoC Design Conference (ISOCC)*, 2010, pp. 440–444.

[15] N. Alfaraj *et al.*, "HOPE: Hotspot congestion control for clos network on chip," in *International Symposium on Networks-on-Chip (NOCS)*, 2011, pp. 17–24.

[16] E. Kakoulli *et al.*, "HPRA: A pro-active hotspot-preventive high-performance routing algorithm for networks-on-chips," in *International Conference on Computer Design (ICCD)*, 2012, pp. 249–255.

[17] D. Xiang *et al.*, "Thermal-aware test scheduling for noc-based 3D integrated circuits," in *International Conference on Very Large Scale Integration (VLSI-SoC)*, 2013, pp. 96–101.

[18] J. Murray *et al.*, "Thermal hotspot reduction in mm-wave wireless noc architectures," in *International Symposium on Quality Electronic Design (ISQED)*, 2014, pp. 645–652.

[19] D. Zhu *et al.*, "TAPP: Temperature-aware application mapping for noc-based many-core processors," in *Design, Automation and Test in Europe (DATE)*, 2015, pp. 1241–1244.

[20] M. F. Reza *et al.*, "Task-resource co-allocation for hotspot minimization in heterogeneous many-core nocs," in *International Great Lakes Symposium on VLSI (GLSVLSI)*, 2016, pp. 137–140.

[21] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 86–96.

[22] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Computer Architecture News (CAN)*, vol. 39, no. 2, pp. 1–7, 2011.