Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

Application Aware Workload Allocation To Optimize The Performance In Network On Chip Based Manycore Processor

D. Radha[1*] , Mushtaq Shaikh[2] , Vamsi Silla[2] , Minal Moharir[3]

[1]Department of Computer Science & Engineering, Amrita School of Engineering Bengaluru, Amrita Vishwa Vidyapeetham, India.

[2]Department of Electronics and Communication Engineering, RV College of Engineering, Bengaluru, India.

[2]Department of Electronics and Communication Engineering, RV College of Engineering, Bengaluru, India.

[3]Department of Computer Science and Engineering, RV College of Engineering, Bengaluru, India.

Abstract

With the advancement of powerful standalone applications, web applications, mobile applications etc., the demand for multi-core architecture has increased exponentially. In addition, the advances in CMOS technology led today's chip manufacturers to increase the number of processing cores on a chip to improve the overall computation performance. The execution of any application by many cores of the system, may need to communicate with each other during their cache misses. The Network-on-Chip (NOC) based architecture inherently augments the communication between different cores on a chip. Various routing algorithms can facilitate this communication among the cores on NOC. The proposed method focuses on assigning the same application's instructions to the nearby cores, and as the application is executed within nearby cores, the communication required is within those cores only, which in turn reduces the latency. Adaptive Routing algorithms enhances the performance of communication with less latency. The comparison is carried out on parameters such as average flit latency, average packet latency, total energy, and total average power for three routing algorithms: Mesh _ XY, Odd-Even, Adaptive Odd-Even in a 2-d Mesh topology. These algorithms are implemented and tested in the Gem5 Simulation tool.

With the help of SPEC CPU 2006 benchmarks, the experimental study shows that the performance parameters are optimized when an entire application is executed on one quadrant of a topology compared to the random execution of an application's thread on various cores, which are far from each other.

Keywords: Manycore architecture, Network on Chip, Adaptive routing algorithms, Latency, Power, Application

Introduction

In today's world, to address real-world problems, the demand for sophisticated software applications is increased. To facilitate such a higher level of complex programs, there is a need for many electronic hardware blocks that can speed up the processing and reduce the overall computation time. Such growth in the software industry pushed the manufacturers to build multi-core architecture. In the Multi-Core architecture, there are many cores on a chip. Every core on a chip has a cache to further ramp up the processing without accessing the main memory every time. So, these vast applications will run on the multi-core, and better performance will be needed. Therefore, to increase the performance capability of the chip, the advent of NOC started.

In the NOC context, every core on the chip will communicate with others to increase the performance. The communication happens between them through packet transferring mechanism. It is analogous with the real-world networking, where every device on the internet is connected to some networking device where the routing or the switching happens. The end device has its address based on which the transfer of packet occurs. Similarly, on a chip, every core is interconnected to a router, which further drives towards the destination with the help of a routing algorithm. The routing algorithms are divided into two broad sets, namely, deterministic, and adaptive. An example of a deterministic routing algorithm is dimension order routing. This algorithm precomputes the route from source to destination and obtusely routes all the packets in the same path. Algorithms like these won't consider the network's congestion state, due to which it takes a lot of time to traverse [1]. This algorithm is reliable and has low latency in a congestion-free network. However, on the other hand, another type, i.e., adaptive routing algorithm, considers the congestion rate at the routers; based on some intelligence, it takes an alternative path than the congested route. This intelligence-based routing algorithm helps the packets dynamically switch from one port to other.

This paper discusses the adaptive odd-even routing algorithm based on the router delay, which was computed dynamically at every router. Such an algorithm reduces the overall computation power, latency, energy on the larger metric dimensions of Mesh topology. The performance metrics can be even improved based on the execution of a binary on the core. Different threads of a binary file can be executed on a chip randomly at any core far from each other, thereby increasing the cost, as it must traverse to that core for packet transfer. However, the performance metrics cost can be optimized if all the related threads of a binary file are executed in a quadrant of cores on a chip. Since the cores within a quadrant are nearby to each other, it decreases the computational power,

latency etc. This paper even discusses how the spatial distribution of a binaries execution for different routing algorithms and different instruction rates differ. The tool we used to substantiate this research is gem5, where the development was compiled and executed in SE mode.

GEM5 Simulation Tool

Many open-source processor level simulators can simulate the multi-core architecture for multiple ISA's. Out of which, GEM5 is the most comprehensive simulation tool. The GEM5 simulator is written in python and C++. This tool runs in either Full System Mode (FS) or System Emulation Call Mode (SE). In the System Call Emulation Mode, the user-level space is emulated; however, in the Full System Mode, it emulates the entire machine, including the operating system and the number of cores present. In addition, the tool consists of various configuration files, which provides many options that can be executed. Enhancement of the tool is also feasible by adding multiple user-defined classes in C++ to fulfil the user's requirement. Furthermore, it is possible to run SPEC CPU 2006 benchmarks and the tool encompass various default standard benchmarks such as namd, leslie etc., that give an option to run the benchmark. Finally, after execution, all the output parameters like latency, power, energy etc., are written in a statistics file. Because of its extensive features, this research work is implemented on the GEM5 simulation tool.

Related Works

While the router implements some routing algorithm, there are some restrictions on the path the packet traverses. In the odd-even routing algorithm, for the packet, some 90-degree turns are prohibited at the router, because of which the system performance is affected by the number of restricted turns. As the name suggests, the packets are initially routed along X-direction and then along Y-direction in the XY routing algorithm. Similarly, in the Odd-Even routing algorithm, there are few restrictions. For example, an Eastbound packet cannot take the North/South direction at the even column router [2]. Likewise, a North/Southbound packet cannot take a West direction at the odd column router by the same token. Such restrictions are necessary to avoid deadlocks.

Various approaches towards the optimization of the routing path can be achieved on different topologies using different routing algorithms and traffics. Some of them are repetitive turn model for adaptive routing [3] their “goal is to exploit the designing space for logic-based routing algorithms and proposed new logic-based routing algorithms that outperformed the state-of-the-art counterparts”. The work on “The Odd-Even turn model for adaptive routing” presents a model without virtual channels for designing adaptive wormhole routing algorithms for mesh. When Xinghua Tang, Chunhui Wu [4] conducted a case study of an adaptive routing algorithm for NoC, known as Odd-Even turn model, simulation results conclude that in local region there is a high amount of congestion compared to outer regions. It suggests that this issue must be taken into consideration while designing routing algorithm by the researchers.

As the single processor has limitations, multi-core processors are the substitutes that reduce energy, power consumption, and increase computational speed. The software is designed to execute multiple threads of the same process on different cores of a chip. If the code is not threaded, then the entire process executes on a single chip's core, leaving other cores idle. The proposed work, “Implementation and Analysis of Adaptive Odd-Even Routing in Booksim 2.0 Simulator”,[4] presents a deadlock-free adaptive odd-even routing (OE) to route-cache miss packets in the cores with less delay in a 2D mesh topology. The algorithm is implemented and tested in Book sim 2.0 simulator. The paper shows the Simulation result shows the analysis of the routing algorithm in various synthetic traffic patterns.

When the processor serves a task, it should make sure that the process is executed before the deadline restriction. These are hard real-time systems where the operating system should schedule the process before the deadline of the process occurs.[5] During these rigid time scenarios, the NOC architecture processes swiftly as it does not depend on the main memory for most of the time. Furthermore, the NOC implements adaptive algorithms to address rigid real-time systems, which considers congestion before the route is computed.

The proposed work emphasizes on the congestion and implements a fully adaptive Odd-Even delay-based routing algorithm where the binaries are executed within a quadrant for various algorithms in the GEM5 simulation tool for different instruction rates.

System Development

Delay Based Adaptive Odd Even Routing

In a 2-D Mesh topology, several routing algorithms can effectively route the packet from source to destination. The routing algorithms are broadly classified into two categories: Deterministic and Adaptive. Deterministic Algorithms have a predefined path from source to destination. The algorithm does not consider the ever-changing parameters in the network and follows a precalculated path. Contrary to this, adaptive algorithms effectively compute their path from the source node to the destination node. In an adaptive algorithm, the packet selects the most effective path amongst all available paths at every node. The best example of an adaptive routing algorithm is the Adaptive Odd-Even delay-based Routing algorithm. In this algorithm, the path which has the least congestion is selected at every node. The below algorithm is used to implement the routing in the Gem5 tool

.

```
Algorithm 1: ODD EVEN ALGORITHM IN GEM5
available paths = NULL;
Ex = Dx-Cx;
Ey = Dy-Cy;
initialization;
if Ex = 0 then
    if Ey > 0 then
        Append North to Available Paths;
    else
        Append South to Available Paths;
    end
end
if Ex > 0 then
    if Ey = 0 then
        Append East to Available Paths;
    else
        if Cx is odd or Cx = Sx then
            if Ey > 0 then
                Append North to Available Paths;
            else
                Append South to Available Paths;
            end
        end
        if Dx is Odd or Ex! = 1 then
            Append East to Available Paths;
        end
    end
else
    Append West to Available Paths;
    if Cx is Even then
        Append North to Available Paths;
    else
        Append South to Available Paths;
    end
end
```

Fig. 1 Odd Even Algorithm in Gem5

Using Algorithm 1, depicted in Fig.1, the available path for a packet at a Node is determined. The delay-based decision algorithm can be added as a plugin to algorithm 1 to compute the effective route. The delay-based algorithm is used to compute the maximum delay of each router. A typical three-stage pipelined No C router architecture is shown in Fig. 2. Every input port has a FIFO-based input buffer, which can be seen as a single virtual channel used to hold blocked flits. The routing computation (RC) module sends a channel request signal to the switch allocator (SA) for data in each input buffer. If the downstream buffer at a neighboring router has vacant space, SA will allocate the channel and route the data flits through the crossbar switch toward the designated downstream router at the switch traversal (ST) stage. However, a channel is owned by a packet, but buffers are allocated on a flit-by-flit basis. As such, an idle packet may continue block a channel even when another packet is ready to use the same channel, leading to inefficient resource utilization. This introduces a delay in each router. An algorithm is modelled which calculates the maximum delay of each router every time a flit passes through it. The maximum router delay of each router is stored which gets updated dynamically throughout the lifecycle of communication.

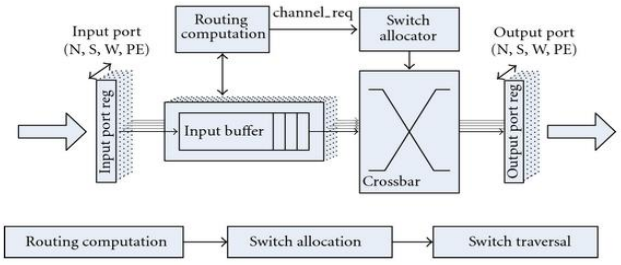

Fig. 2 Block diagram of a NoC Router [11]

Let router delay of router 1 at an instance 0 be $RD1_{0.}$

The max delay of router 1 be $RD1_{max.}$

At tick time = 0, $RD1_{max} = RD1_0$

At tick time = N, delay of router 1 = $RD1_N$

Max Delay at tick time = N; max ($RD1_{max,}$ $RD1_{N)}$

The delay of each router is calculated as the difference between tick times at Input Port and Crossbar switch allocator, the end stage in a typical router block.

Therefore, $RD1_N$ = Tc (Crossbar switch allocator) – Ti (Input Port), where Tc is the simulation cycle tick when the flit is at Crossbar switch allocator & Ti is the simulation cycle tick when the flit enters the Input port.

Using the information of router delays, the Algorithm 2 can now be plugged in Algorithm 1 to efficiently compute the least congested path at a node.

Algorithm 2: Delay-Based Congestion Model

The **"available_paths"** computed in algorithm 1 is used to get a set of available paths for a packet at Node C (Cx, Cy). Corresponding routers associated with the available paths is determined and their respective router id's is stored as depicted in Fig. 3.

```
initialization;
while auto & path: Available Paths do
    if Path = North then
        path _ id = id + num _ cols;
        available _ path _ id.append(path_ id);
    end
    if Path = South then
        path _ id = id - num _ cols;
        available _ path _ id.append(path_ id);
    end
    if Path = East then
        path _ id = id + 1;
        available _ path _ id.append(path_ id);
    end
    if Path = West then
        path _ id = id - 1;
        available _ path _ id.append(path_ id);
    end
end
while auto & path: Available Paths do
    latency = router_ id_ to_ delay [id];
    if latency < lowest_ latency then
        lowest_ latency = latency;
        lowest_ latency_ id = id;
    end
end
```

Fig. 3 Delay Based Decision Making Algorithm

Once the available_path_ids are computed, their corresponding max router delay is determined and the path with least delay is selected to route the packet at Node C (Cx, Cy). The lowest_latency_id is the path with the least congestion, and it is assigned to the packet at Node C (Cx, Cy). Consider an example shown in Fig. 4 where source node is 0 and destination node is 15.

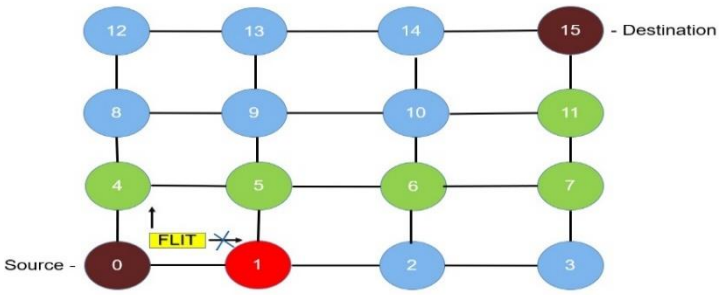

Fig. 4 Flit traversal for delay-based adaptive odd even algorithm.

The packet at node 0 has 2 available paths for traversal. Based on the delay-based algorithm, the router delay of adjacent node ‘4’ is less than node ‘1’. So, the flit moves to node ‘4’. Similarly at node ‘4’ the algorithm computes the available_paths. From the available paths, it selects adjacent node ‘5’ which has less router delay compared to node ‘8’. Subsequently, the same algorithm is applied at each node till the flit reaches the destination node.

Application Based Workload Allocation

Multi-Core Processors are those which have multiple cores on a single integrated chip. Because of the demand for computational capacity, the necessity to ramp up the processing speed without compromising the chip's area was challenging for the chip manufacturers. To address this necessity, a group of experts came up with Network-on-Chip. The miniature NOC architecture is analogous to macro-level computer networking, consisting of routers, switches, etc., where all the routers are interconnected. The blend of processors such as GPU, DSP, Low Power consumption processors etc., are integrated on a single chip. Thus, while a high computational algorithm's process is being executed on the chip, the interaction with the memory controller for data or instruction access has to be reduced. Because of which every core on the chip has its own Instruction-Cache and Data-Cache to lessen the processing latency. In real-world scenarios, in critical situations, the cores on the chip must serve the task in a brief period.

For this reason, the processes are categorized into broadly two sets: hard real-time processes, soft real-time processes. The hard real-time processes or events should be executed before a deadline system's tick; otherwise, the control system will lead to a catastrophe. During the execution of a vast process, multiple cores execute the different threads of the program and interact with each other to swiftly run the program. In this paper, with the help of SPEC benchmark programs, the binaries are explicitly made executed on a particular core of the chip by running the python configuration files. The default benchmark binaries were:

- NAMD: This program comes under molecular biology, Classical Molecular Dynamics subject. The software is implemented in the C++ programming language. The GEM5 tool has input. namd configuration file, where all the related parameters can be configurable. By default, the code is run for 38 iterations.
- LBM: This program comes under fluid dynamics category. The software implements "Lattice Boltzmann Method" (LBM), which simulates the incompressible fluids. The software implements the "Lattice Boltzmann Method" (LBM), which simulates the incompressible fluids. In the Lattice Boltzmann Method, a steady-state solution is accomplished by running an exact number of model time steps. For example, 3000-time measures are computed for the reference workload, while a far smaller number of time steps are computed for the test and training workloads. The software is implemented in the ANSI C programming language.
- LESLIE3D (Large-Eddy Simulations with Linear-Eddy Model in 3D): This program comes under computational fluid dynamics. The software is implemented in the Fortran programming language. This benchmark consists of huge derivative mathematical code that validates the core standard. It consists of various sizes, which can be configured in the input configuration file.
- BZIP2: This program comes under compression of the package. The software is implemented in the ANSI C programming language.

These default benchmark programs mentioned above are written in multiple programming languages, which helps to validate the performance of the NOC architecture. The deterministic and adaptive routing algorithms discussed so far facilitates the communication between multiple cores where multiple threads of these above images are being executed randomly. Due to this randomness of execution, the performance metrics such as latency, power, energy etc., are not optimum. However, these metrics can be optimized if an execution of a binary file can be confined within a quadrant as shown in the Fig. 5.

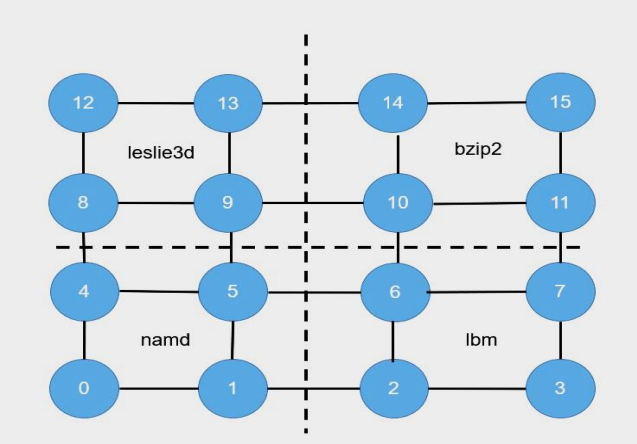

Fig. 5 Application based 4x4 2-d Mesh topology

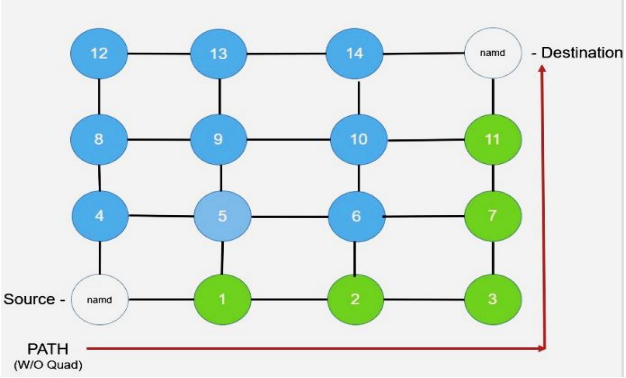

Fig. 6 Packet traversal in a random resource allocation network

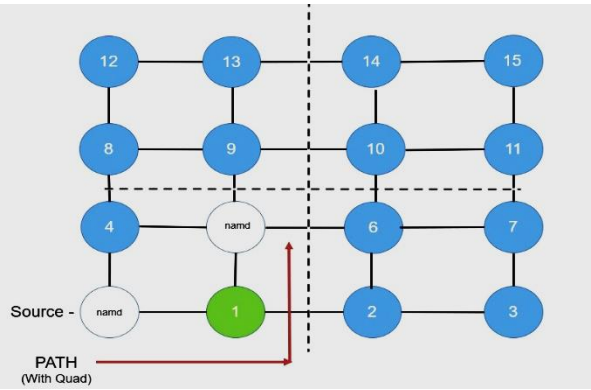

Fig. 7 Packet traversal in an application aware resource allocation network

Consider an instance of a binary file- ‘namd’ executing on a 4x4 2-d Mesh topology. Using Spec-2006 CPU Benchmark, the ‘namd’ binary is forced to execute on core 0 and core 15 as shown in Fig. 6. Here, source node is 0 and destination node is 15 for a packet. The packet must traverse through the path using a routing algorithm. The source and destination node are relatively far from each other, thus increasing the latency, power etc. However, these (performance metrics) can be optimized if the binaries execute within a quadrant, as shown in Fig. 7. Hence, when a core needs

to communicate with other cores to exchange significant information, communication happens within the quadrant, thereby substantially improving the latency and other metrics.

This paper implements an application aware workload allocation network using SPEC-2006 CPU benchmark for various routing algorithms, namely – 'Mesh _ XY', 'Odd Even', 'Delay-Based Adaptive Odd Even' on Gem5 simulation tool. Intrinsically, the tool supports the 'Mesh _ XY' routing algorithm only. It was further enhanced to support adaptive algorithms discussed above. Manual parameters were taken from the user and based on the input; the required routing algorithm was executed in the backend. Here, instruction count plays a vital role in determining the performance criterion of a routing algorithm. Therefore, the routing algorithms were compared for numerous instruction cycles – [500, 1000, 1500, 2000]. This work was further extended to an application aware quadrant-based network, and comparative results were analyzed among various routing algorithms and instruction counts. Similarly, the procedure was mirrored for an 8x8 2-d Mesh topology.

Results and Analysis

The proposed work is carried out on GEM5 simulation tool in System call Emulation (SE) mode. The implementation is carried out for two mesh sizes: 4x4 and 8x8 2-d Mesh topology. For each mesh size, system parameters namely, Average Flit Latency, Average Packet Latency, Total Average Power and Total Energy are compared using three routing algorithms which are 'Mesh _ XY', 'Odd Even' and 'Delay-Based Adaptive Odd Even'. This comparison is performed with and without application aware allocation i.e., when binaries are executed within the quadrant vs random execution of binaries.

Fig. 9 and Table 1 denotes the comparison of Average Flit latency. It is evident from the graph that, allocation of binaries within a quadrant outperforms the random execution of binaries for all the routing algorithms with maximum difference of 0.163 ticks. The adaptive odd even routing algorithm shows the lowest flit latency of 14.345 ticks for IC of 2000 with quadrant.

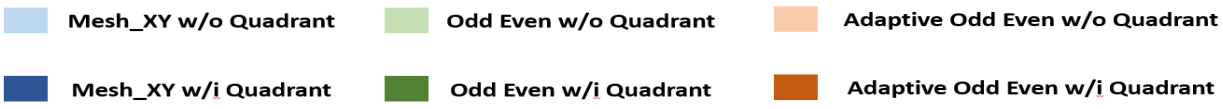

Fig. 8 Graph Legend

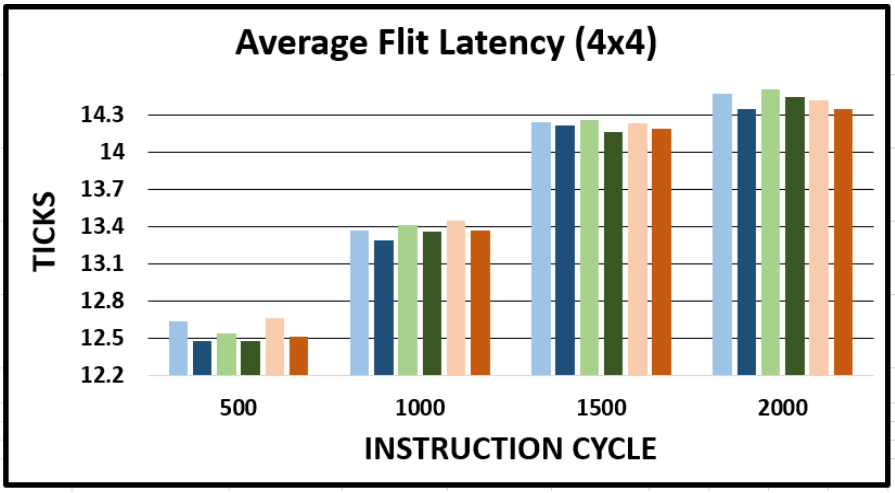

Fig. 9 Bar Graph depicting Average Flit Latency for 4x4 mesh topology

Instruction Count	Average Flit Latency (4x4)					
	Mesh _ XY		Odd Even		Adaptive Odd Even	
	W/o Quad	W/i Quad	W/o Quad	W/i Quad	W/o Quad	W/i Quad
500	12.638	12.475	12.54	12.478	12.661	12.512
1000	13.369	13.29	13.414	13.36	13.444	13.37
1500	14.273	14.213	14.254	14.162	14.227	14.185
2000	14.472	14.346	14.507	14.444	14.413	14.345

Table. 1 Comparison of Average Flit Latency for 4x4 mesh topology

Similarly, average packet latency for a 4x4 2-d Mesh topology is compared in Table 2. The bar graph depicted in Fig.10 shows that the allocation comparatively has lower latency with adaptive odd even showing the least flit latency of 16.646 ticks for 2000 instruction cycles.

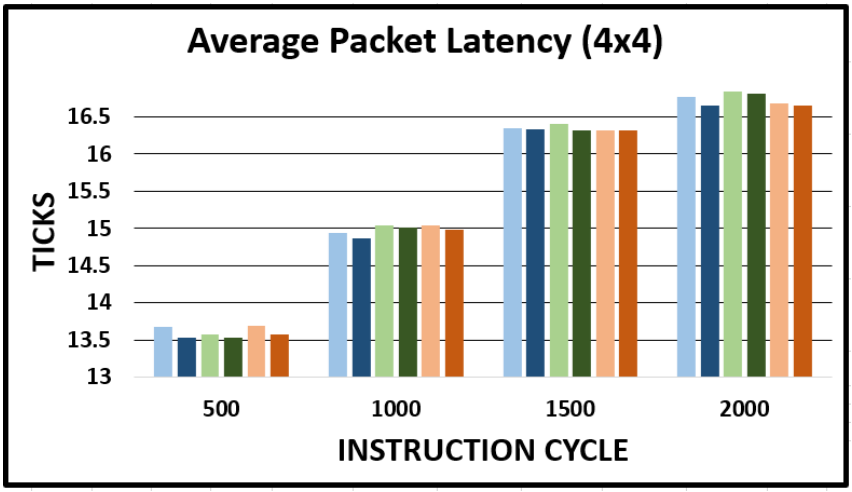

Table. 2 Comparison of Average Packet Latency for 4x4 mesh topology

Instruction Count	Total Energy (m J) (4x4)					
	Mesh _ XY		Odd Even		Adaptive Odd Even	
	W/o Quad	W/i Quad	W/o Quad	W/i Quad	W/o Quad	W/i Quad
500	0.073	0.072	0.073	0.073	0.074	0.073
1000	0.177	0.177	0.181	0.178	0.181	0.177
1500	0.229	0.229	0.233	0.231	0.230	0.229
2000	0.287	0.283	0.288	0.289	0.281	0.280

Fig. 10 Average Packet Latency for 4x4 mesh topology

In an application aware allocation, the packets communicate within a quadrant, thereby the no of hops that each packet traverses are significantly less compared to random allocation. Therefore, the total energy and average power consumption is less for the overall network. Table 3 compares the total energy consumed by the network for various routing algorithms. It is evident from Fig. 11 that, the total energy consumed by the network with quadrants is less for each of the tested routing algorithms. With adaptive odd even routing algorithm, the network consumes the least energy of 0.280 m J for 2000 instruction cycles with application aware allocation.

Table. 3 Comparison of Total Energy (m J) for 4x4 mesh topology

Instruction Count	Average Packet Latency (4x4)					
	Mesh _ XY		Odd Even		Adaptive Odd Even	
	W/o Quad	W/i Quad	W/o Quad	W/i Quad	W/o Quad	W/i Quad
500	13.684	13.53	13.573	13.534	13.69	13.571
1000	14.944	14.873	15.045	15.011	15.039	14.98
1500	16.35	16.337	16.408	16.32	16.322	16.314
2000	16.772	16.652	16.836	16.82	16.68	16.646

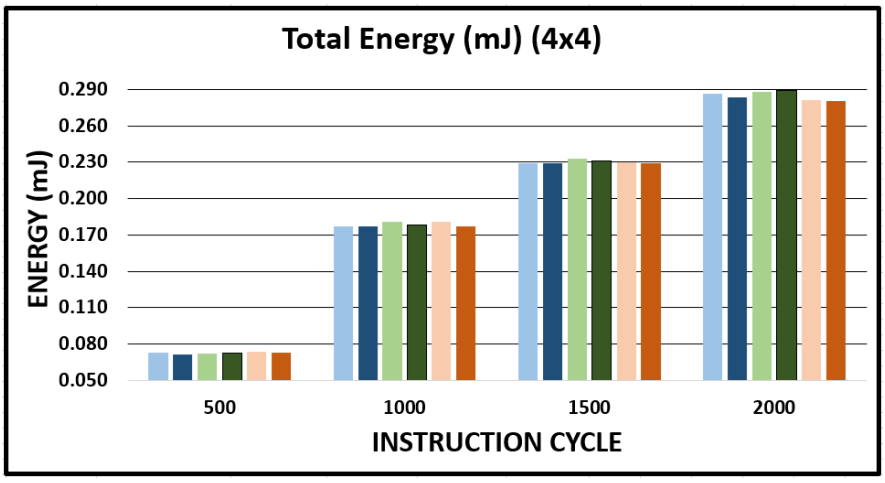

Fig. 11 Total Energy for 4x4 mesh topology

Similarly, Table 4 compares the Total Average Power for the network. The total average power depicted in Fig. 12 elucidates that, the least power is consumed by an application aware network based on quadrants for adaptive algorithm with the minimum value of 1.279 m W.

Table. 4 Comparison of Total Average Power (W) for 4x4 mesh topology

Instruction Count	Total Average Power (W) (4x4)					
	Mesh _ XY		Odd Even		Adaptive Odd Even	
	W/o Quad	W/i Quad	W/o Quad	W/i Quad	W/o Quad	W/i Quad
500	1.202	1.205	1.207	1.205	1.204	1.204
1000	1.355	1.358	1.361	1.363	1.359	1.356
1500	1.321	1.327	1.318	1.317	1.324	1.315
2000	1.280	1.288	1.283	1.283	1.287	1.279

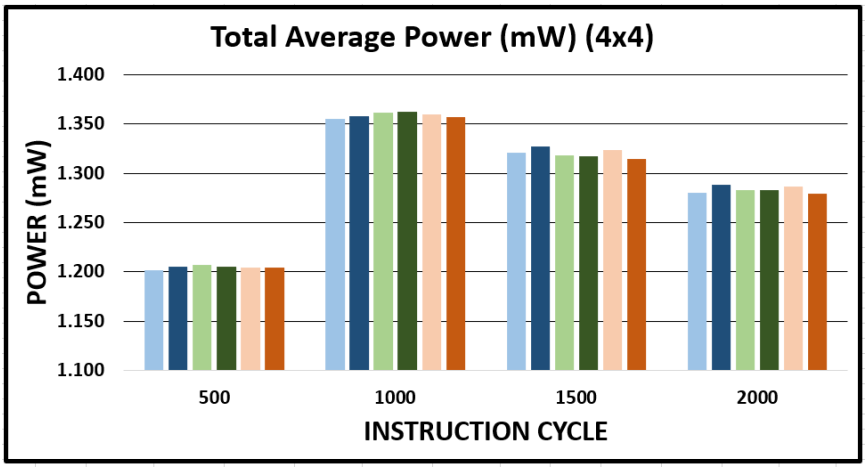

Fig. 12 Total Average Power for 4x4 mesh topology

Likewise, the implementation is extended to 8x8 2-d mesh topology. The results of 8x8 conforms with 4x4 mesh size. In Fig. 13, Average Flit latency for an 8x8 mesh topology is compared. It's evident from the graph that, Adaptive Odd Even routing algorithm with application aware allocation shows least latency of 20.278 ticks for 2000 Instruction cycles.

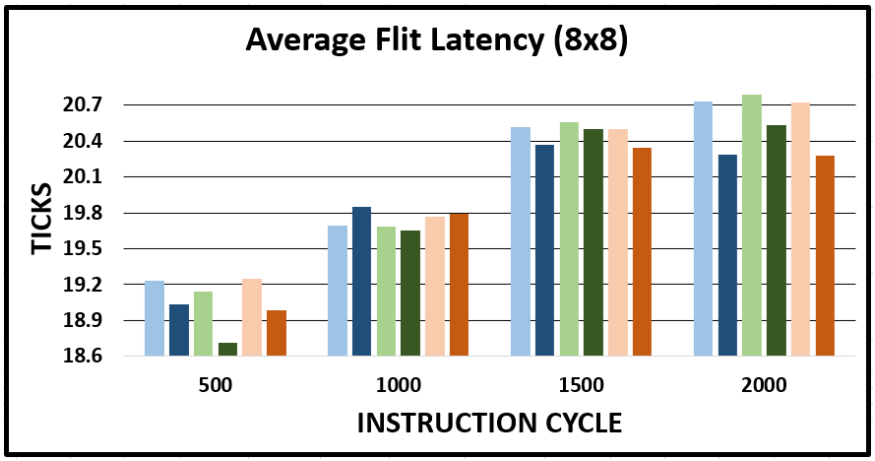

Fig. 13 Average Flit Latency for 8x8 mesh topology

Fig. 14 compares the Average Packet Latency for various routing algorithms which are implemented with and without application-based allocation strategy. Quadrant-based adaptive odd even routing algorithm performs best for 2000 Instruction counts having least latency of 22.516 ticks.

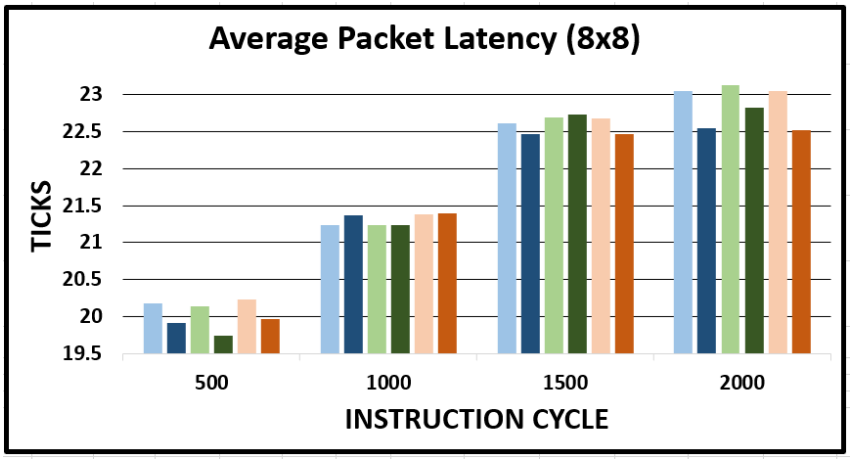

Fig. 14 Average Packet Latency for 8x8 mesh topology

Subsequently, Total Energy and Total Average Power are compared in Fig. 15 and Fig. 16 respectively. For 2000 instruction cycles, Adaptive Odd Even routing algorithm with quadrant-based approach consumes lowest energy of 1.238 m J and utilizes least power of 1.257 m W.

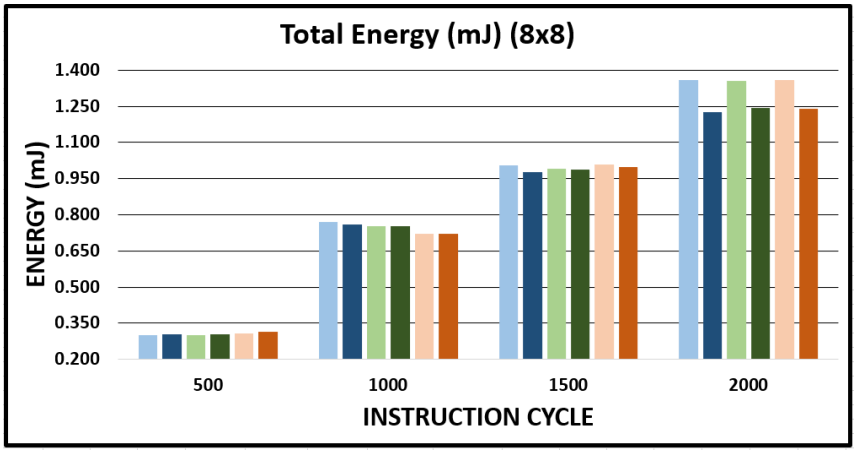

Fig. 15 Total Energy (m J) for 8x8 mesh topology

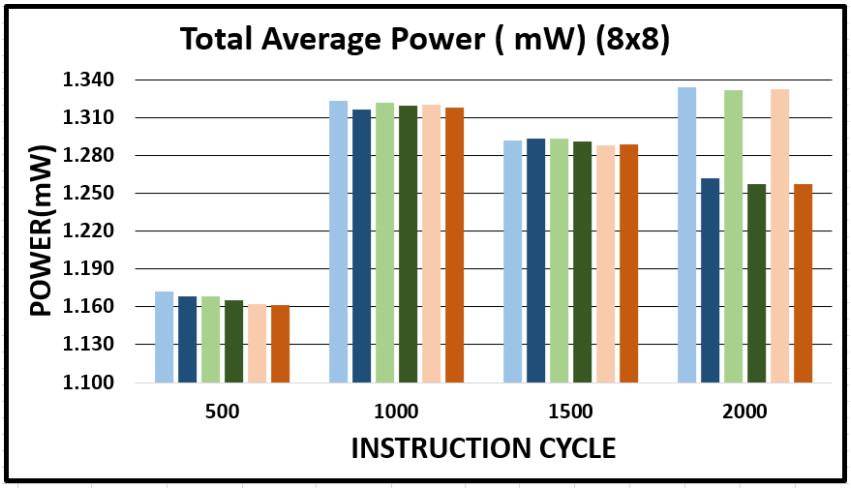

Fig. 16 Total Average Power(m W) for 8x8 mesh topology

Conclusion

When compared to the deterministic routing algorithm, adaptive routing algorithms are efficient in the real Network-on-Chip scenario. Moreover, the intelligence with which the algorithm computes the path considering the router delay as the essential criterion gives the best results for packet latency, power, energy, etc., compared to other routing algorithms. Such an implementation can be executed on mesh topology with different 2d Mesh size. Even the above discussed adaptiveness in the algorithm can be executed on any topology. The implementation is done with the help of the GEM5 simulation tool where we had studied the performance of various system parameters. The statistics file in the GEM5 tool had given clear stats of all the parameters after the execution of the command. The tool was further furnished with the above-mentioned routing algorithms, where the user can manually alter the routing algorithm. The results show that the adaptive odd-even routing algorithm was the best in terms of its performance metrics, where if the mesh size and instruction count increase. Such an algorithm along with the workload allocation,

will overall improve the efficiency and reliability of the system. The algorithm and workload allocation are tested with benchmark files. The base idea of workload allocation can be applied to any topologies depending upon the adjacency of nodes. The routing algorithms with this workload allocation as the base can be further implemented in other topologies like Quad-Tree, Fat tree, Torus, Dragonfly etc.

References

[1] Minghua Tang, Chunhui WuA "Case Study of the Odd-Even Turn Model", Published in 2nd International Conference on Consumer Electronics, Communications and Networks (CEC Net) 2012.

[2] Minghua Tang, Xiaola Lin, and Maurizio Palesi "The Repetitive Turn Model for Adaptive Routing", IEEE Transactions on Computers, Vol. 66, No. 1, January 2017

[3] Terrence Mak, Peter Y. K. Cheung, Kai-Pui Lam, and Wayne Luk" Adaptive Routing in Network-on-Chips Using a Dynamic-Programming Network", IEEE Transactions on Industrial Electronics (volume 58, issue:8,August 2011.

[4] Maddula N V Sesha Sai Teja, K Sai Sumanth Reddy, D Radha, Minal Moharir, "Multicore Architecture and Network on Chip: Applications and Challenges", presented at ICIC 2018, Amrita School of Engineering, Bengaluru. India.

[5]Nallasamy Viswanathan, Kuppusamy Paramasivam, and Dr. Somasundaram K., "Vertical Links Minimized 3D NoC Topology and Router Arbiter Design", International Arab Journal of Information Technology, vol. 15, pp. 469–478, 2018

[6] M. Vinodhini, N.S Murty, "Reliable Low Power NoC Interconnect", Microprocessors and Microsystems, vol. 57, pp.15-22, 2018.

[7]Seema , Pawan Kumar Dahiya "Network-on-Chip: A State-of-the-art Review", IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) Volume 7, Issue- 4, Ver. I (Jul. - Aug. 2017), PP 29-35.

[8] Wang Zhang, Ligang Hou, Jinhui Wang, Shuqin Geng, Wuchen Wu," Comparison Research between XY and Odd-Even Routing Algorithm of a 2- Dimension 3X3 Mesh Topology Network-on-Chip", Published in 2009 WRI Global Congress on Intelligent Systems.
[9]Ge-Ming Chiu, "The Odd-Even Turn Model for Adaptive Routing", Published in IEEE Transactions
on Parallel and Distributed Systems(Volume: 11 , Issue: 7 , Jul 2000).

[10] Mohammad Sadrosadati, Ramin Bashizade, Ali Shafie and Hamid Sarbazi “A Method to Improve
Adaptivity of Odd-even Routing Algorithm in Mesh NoCs”,Published in 24th Euro micro International
Conference on Parallel, Distributed, and Network Based Processing 2016.
[11] Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, Sao-Jie Chen, "Networks on Chips: Structure and Design Methodologies", Journal of Electrical and Computer Engineering, vol. 2012, Article ID 509465, 15 pages, 2012. https://doi.org/10.1155/2012/509465