A New Hierarchical Interconnection Network with considering Efficient Energy Usage for Exa-scale Supercomputing

By Faiz Al Faisal

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Doctorate of Information Science
Postgraduate Program in Information Science

Written under the direction of
Professor Yasushi Inoguchi

September, 2018

Abstract

The requirement of high processing power is enormous; one of the high desire of next generation HPC systems. Since every computer chip has limited processing power, sequential processors can't be the suitable choice. For example, an Intel Core i7-3630QM processor (4 cores, 22nm fabrication process) can achieve about 76.8GFlops through the requirement of 45W electrical power usage. However, the requirement for exa-scale computing will require about 13 million of connecting such processors. Todays most powerful supercomputer Sunway Taihulight System has already achieved about 93 petaFlops performance with 10,649,600 cores requiring about 15.3MW electrical power using low degree 2DMesh interconnect (2DMesh has the performance constraints and faster saturation rate). Moreover, a high degree network like- Tofu (6DMesh/Torus) interconnect used in K-computer has achieved 10.51 petaFlops performance (88,128 SPARC64 VIIIfx processors, Tofu interconnect with 10 cores to cores connectivity) by requiring 12.6MW of electrical power. Hence, to build an exa-flops Tofu system, it will require near about 1260MW of electrical power with the current advancements. These observations confirm that conventional structures are not feasible for the next generation networks due to the high power usage for the high degree core to core connectivity (Tofu interconnect) and also shows poor network performances (2DMesh interconnect). Hence, the possible solution to reach the next generation exa-scale performance is to redesign the "Interconnection Network".

Exa-scale supercomputing requires network scalability over millions of cores and the performance constraint affects heavily for the large system along with the total power usage. In considering those constraints our focus resides on the "Hierarchical Interconnection Networks (HIN)". HINs possess the features like- constant node degree, small average distance, better bisection width, small number of wires and low network latency with high throughput. Constant node degree ensures the fixed router cost throughout the entire system, small average distance eliminates the possibility of the performance degradation, better bisection width ensures the network traffic handling capability, wiring complexity is effective for reducing the network power usage and finally, network latency ensures the packet reachability with the requested traffic load. This research also considers a new parameter of "Network Energy Usage" to ensure the high performance and the low power usage. Moreover, we have considered two possible network configuration of 65K cores and 1M cores analysis to ensure the superiority of our network for the exa-scale system.

Keywords: Interconnection Network, Hierarchical Flattened Butterfly Network, Estimation of Power Consumption, Dynamic Communication Performance, Energy Usage.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Exa-scale performance is the next goal post for next generation supercomputers and most likely next generation high performance computing is solely depends on the massively parallel computers. Modern massively parallel computer (MPC) systems like- K-computer has already achieved 10.51 petaFlops performance with more than 80,000 computing cores and also requires about 12.6MW electrical power [1]. In addition, to achieve the exa-scale performance the scaling will be required about 95 times and required power consumption will be close to 1200MW for K-computer. On the other hand, Blue Gene/Q supercomputer requires 6.6MW of electrical power in achieving 20 petaFlops performance with 1.57M processor core. However, this supercomputer will require about 330MW of electrical power for building the exa-scale system. Therefore, the reduction of the power usage of the MPC systems is the one of the important issues for the next generation exa-scale supercomputers along with continuing the other constraints like- low network performance, poor scalability, poor throughput and large network latency [2].

The overall performances as well as the power consumption of MPC systems are heavily affected by the interconnection networks and its processing cores. Interconnection network acts as a communicating path for processing cores and as well as for the memory units. The main goal of this research is to find a suitable hierarchical interconnection network that will be suitable for the exa-scale processing. As the processing power of the single CPU core is limited, it is important to integrate a large number of processing cores to enhance the performance of the massively parallel system. Titan, which already achieved 17.59 petaFlops speeds using about 560,640 processor cores, including 261,632 NVDIA K20x accelerator cores. Hence, to achieve the exa-scale computing we need to find a suitable interconnection network that have high scalability with shorter global interconnect for the low power usages. Consequently, every MPC system requires interconnection network as an obvious choice. Figure 1 shows the latest scenario of existing interconnects for MPC systems, according to system shares [3]. This figure confirms that the vastly used network is the Tree network for the MPC systems, which was used in early days and also it has has a big concern in case of network performance and even most of the modern

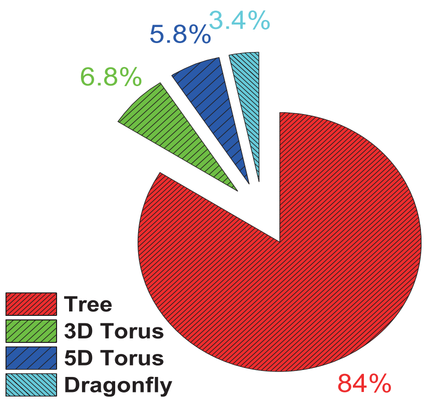


Figure 1.1: Existing network topology for system shares [3]

supercomputers are build upon the conventional Mesh and Torus topology (for example- one of the top supercomputer, Sunway TaihuLight uses the 2DMesh network as their network interconnect [4]). In MPC systems, the total number of outgoing links like- on-chip as well as off-chip links is a big concern, due to power usages as well as high latency. However, Friedman shows 3D NoC requires less power usage than the 2D NoCs with shorter vertical links [5]. Hence, the interconnect pattern for network topology is a vital issue for the next generation exa-scale supercomputers.

On the other hand, efficient energy usage means delivering the same outcome while using less energy. Todays most of the power planets produces the electrical power through burning coal, fuel and gas. And burning them results in the conversion of carbon to carbon dioxide (CO_2), which is then released into the atmosphere. The estimated CO_2 emission from burning fossil fuel is about 10 billion tonnes yearly. This results in an increase in the earths level of atmospheric CO_2, which enhances the greenhouse effect and contributes to global warming. On the other hand, the amount of heat generated from the electronic devices is equivalent to the power input. For example- a heatsink rated at 10^0C/W will get 10^0C hotter than the surrounding air when it dissipates 1 Watt of heat. Hence, the reduction in energy usage ensures the reduction of impacts on the environment and even low power usages ensures the less failure rate for the devices.

1.2 Problem Statement

Networks are the only way for the processors to be interconnected in a MPC system. High degree networks (4D, 5D) ensure maximum performance through a large number of core-core interconnections. However, high radix or node degree introduces heavy power usage. Hence, the problem for next generation MPCs will be the maximum performance through low power usage which leads to the efficient network. On the other hand, 3D NoCs are one of the attractive feature for the exa-scale systems for the low power usages (decrease 62%) than the 2D NoCs [5]. However, 3D NoCs have not been available in the market due to the high fault tolerant issues of through-silicon via (TSV) links and its massive heat generation. Even today's modern supercomputers have also been classified with green list [6]. Table 1.1 shows the recent green listed supercomputers, where the recent MPC systems as Gyoukou ensures the maximum petaFlops/megawatt efficiency of 14.14 (about 42.88% better than the Piz Diant).

On the other hand, exa-scale system is the only desire for the next generation supercomputing. However, exa-scale system can be achieved through interconnecting millions of cores. For example- China's Sunway TaihuLight supercomputer uses 40,960 64-bit RISC processors, containing 256 processing cores for each processor and an additional four auxiliary cores for system control for a total of 10,649,600 CPU cores for the entire system. Sunway system achieved about 93 petaFlops linkpack performance. However, integrating a large number of CPU cores requires huge power usages as well as the network latency trends to get saturated very soon. Sunway system requires about 15MW of electrical power to run the full system and considers the 2DMesh network for the core Networks are the only way for the processors to be interconnected in a MPC system. High degree networks (4D, 5D) ensure maximum performance through a large number of core-core interconnections. However, high radix or node degree introduces heavy power usage. Hence, the problem for next generation MPCs will be the maximum performance through low power usage which leads to the efficient network. On the other hand, 3D NoCs are one of the attractive feature for the exa-scale systems for the low power usages (decrease 62%) than the 2D NoCs [5]. However, 3D NoCs have not been available in the market due to the high fault tolerant issues of through-silicon via (TSV) links and its massive heat generation. Even today's modern supercomputers have also been classified with green list [6]. Table 1.1 shows the recent green listed supercomputers, where the recent MPC systems as Gyoukou ensures the maximum petaFlops/megawatt efficiency of 14.14 (about 42.88% better than the Piz Diant).

On the other hand, exa-scale system is the only desire for the next generation supercomputing. However, exa-scale system can be achieved through interconnecting millions of cores. For example- China's Sunway TaihuLight supercomputer uses 40,960 64-bit RISC processors, containing 256 processing cores for each processor and an additional four auxiliary cores for system control for a total of 10,649,600 CPU cores for the entire system. Sunway system achieved about 93 petaFlops linkpack performance. However, integrating a large number of CPU cores requires huge power usages as well as the network latency trends to get saturated very soon. Sunway system requires about 15MW of electrical power to run the full system and considers the 2DMesh network for the core interconnectivity.

1.3 Objective of this Research

The main target of this research is to find the suitable interconnect for next generation exa-scale supercomputers. The initial target for the next generation interconnection networks is to provide sufficient bandwidth. Bandwidth is a considerable factor for the running applications. Hence, network latency should remain considerable as long as ap-

Table 1.1: petaFlops/megawatt analysis for supercomputers

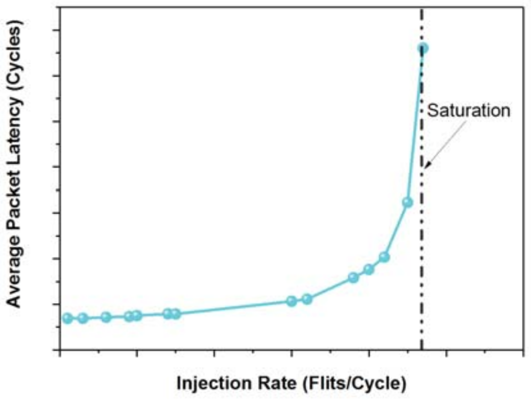
MPC system	Country	Performance (petaFlops)	Power(MW)	petaFlops/MW
Sunway Taihulight	China	93.0	15.4	6.04
Tianhe-2	China	33.9	17.8	1.91
Piz Daint	Switzerland	19.6	2.27	8.63
Gyoukou	Japan	19.1	1.35	14.14

plication bandwidth requirement is much smaller than the bandwidth, which is available through the network. Figure 1.2(a) shows the relation between the delivered bandwidth (offered load) vs. latency, where networks get saturated after a certain amount of delivered bandwidth. On the other hand, figure 1.2(b) shows the relation between delivered bandwidth (throughput) vs. offered bandwidth, where network ensures the maximum amount of traffic the network can handle before it get saturated. Therefore, the initial task of an interconnection network is to transfer the data from the source to destinations with the requirements of following targets:

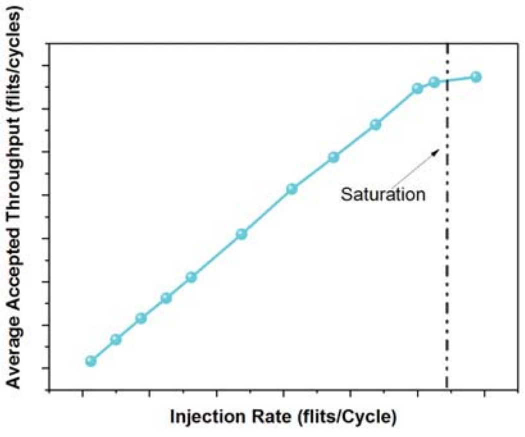
1. Low average transfer time (low latency)
2. High transfer rate (high throughput)
3. Low system cost

One of the most powerful modern supercomputer on earth Sunway TaihuLight System has already achieved about 93 petaFlops performance with 10,649,600 cores requiring about 15.3MW electrical power with the 2DMesh interconnect. The consideration for the 2DMesh network is his constant and low number of out-going links from each core. However, the saturation rate for Mesh network is very fast and even the zero load latency is high. On the other hand, 2D Torus could be an alternative for this latency improvement with increased number of virtual channels, which ensures the less zero load latency than the Mesh networks. However, the increased number of virtual channels induces the increase in power usage due to the activity at the channel buffer. Hence, our designed network considers those observations to reduce the number of virtual channels at the on-chip level and even shows the better zero load latency and high saturation rate than the conventional networks.

Chip Multiprocessors (CMPs) mostly adopt conventional interconnects like- Mesh and Torus, which consume an increasing fraction of the chip power. Moreover, as the technology advances and voltage continues to scale down, static power consumes a large fraction of the total power. Hence, reducing the total power usage is increasingly important for energy proportional computing. Efficient energy usage ensures the reduced amount of energy required to provide the suitable performance. For example- power usage effectiveness (PUE) of the Swiss Supercomputer (CSCS) datacenter prior to 2012 was 1.8, however the current PUE is about 1.25; a factor of 1.5 improvements [7]. With the modern advancements, the biggest concern for supercomputers is the power dissipation. The



(a) Network Latency



(b) Network Throughput

Figure 1.2: Performance Consideration in Latency and throughput

most powerful supercomputer on earth Sunway TaihuLight System has achieved about 93 petaFlops performance with 10,649,600 cores requiring about 15.3 MW electrical power installed with 2DMesh network, which will require 168.3 MW of electrical power for the exa-scale performance (close to 1 nuclear power plant) [4].

In consideration for performance and power usage, we like to introduce our new parameter, which is network energy usage. Network energy is defined by the average transfer time for transmitting the flits from the source core to destination cores (which is also called as the network latency) multiplying with the required network total power usage (this power usage leads by the power requirement from the routers and the connected links). Equation 1.1 shows this assumption condition. On the other hand, the efficiency of energy usage is the reduction of the obtained network energy usage in comparing between two networks with the relative request-probability (r). In dynamic communication performance analysis, packets are transmitted by the request-probability (r) during the simulation clock cycles. This value is calculated in percentage. The lower the network energy usage the better the efficiency can be obtained. As the modern MPCs are highly affected by the power consumptions, efficient energy usage can able to trace the system performance with respect to power usage, which is an absolutely new feature in the field of interconnection networks.

$$\text{Network Energy Usage (NEU)} = \text{Average Transfer Time} \times \text{Network Total Power Usage} \tag{1.1}$$

1.4 Contribution of the Dissertation

Energy consumption completely dominated by costs of data movement. The most critical problem for 3D networks is the massive heat generation. On the other hand, it is obvious that 3D networks require a much higher number of off-chip connection than the 2D networks (50% higher). Even the cost and the power usage for 3D networks is much higher than 2D networks. This consideration leads to 2D NoC architecture is obvious choice for the exa-scale supercomputing. Even the Sunway supercomputer used the 2DMesh network for considering the exa-scale system. However, hierarchical networks are preferable over the conventional networks due to the hierarchical design for the modern MPC systems. Hence, in this research, we are considering a 2D NoC based hierarchical network for the interconnect for next generation exa-scale system.

Packet routing for interconnection networks plays the most crucial role to transmit the messages from one core to another. Bad routing logic can degrade the network performance. On the other hand, dimension-order routing (DOR) has been the most popular for MPC systems due to its minimal hardware requirements and allows the router configuration to simple and cost-effective. The performance of the networks can be found through the dynamic communication performance. Dynamic communication performance can be obtained from the latency and throughput analysis. However, a deadlock-free routing is an obvious choice for any interconnection networks. Our network has also considered a deadlock-free DOR routing for the traffic analysis.

Now, our next phase of research will be evaluating the power usage of the new network. In on-chip level the power consumption of the interconnection network depends on the total router power usages and summation of per link power usages. Even total power consumption highly depends not only on the leakage power, but also the dynamic power of both routers and link modules. On the other hand, the off-chip connection mainly draws the power usage from the leakage power. Hence, our estimation for power usages will require the evaluation of on-chip and off-chip power consumption for various interconnection networks with common parameters. On the other hand, power usage for interconnection network does not reflect the performance analysis. Hence, a network, which shows low power usage with little low performance, has always been neglected due to the poor performance though it could be handy for next generation computers. To sort out this issue, we like to ensure the efficient energy usage for the various interconnects.

Finally, it is being expected from the interconnection network with low cost, low degree, low congestion, high connectivity, high packing density and high fault tolerance. Along with those static parameters, cost-effectiveness factor and also time-cost-effectiveness factor have also been used to evaluate the system cost and system performance of the MPC systems. As the network topology affects the performance metrics, it is important to measure the static network performance for the new interconnection network with comparing against the various networks.

1.5 Organisation of the Thesis

The chapter 1, introduces the requirements for exa-scale supercomputing and their constraints for achieving such requirements. And, now, we would like to ensure the synopsis of the rest of the dissertation as follows:

1. In chapter 2, we considers with most challenging comparators for our network.

Hence, chapter 2 includes the basic architecture of the conventional networks as well as the hierarchical networks. Hierarchical networks treated as the low diameter, low average distance and low cost requirements over the conventional networks. In addition, we shows the requirement of exa-scale supercomputing for the generation HPCs.

2. In chapter 3, we like to introduce our designed network. Our designed network is based on the heterogeneous architecture, which means it flows different architecture at the different network level. In our case, we have considered the 2D flattened butterfly network at the on-chip level and 2DTorus network at the off-chip level or the upper level of network with the certain network configuration.

3. Chapter 4 presents the evaluation of energy usage for various networks. Combining the power usage and the dynamic performance, we obtained the efficient energy usage for our designed network. Those analyses are considered with variable network size like- 65K cores and 1M cores. With the large network size, HINs can show better network performance as well as the low network energy usage.

4. Finally, chapter 5 concludes the thesis with the future work and we have considered the static network performance for our designed network in appendix A, which also ensures the superiority of our designed network over the other networks. Appendix B shows the verification of various simulators that are used in this research. And finally, Appendix C shows the sample code for obtaining the network analysis.

Chapter 2

Related Works on Conventional and Hierarchical Interconnections

2.1 Introduction

Network topology is the most important choice for the MPC systems, where network topology refers as the static arrangement of channels and cores in an interconnection network. Channels are used to transmit the packets over the whole network. In addition, routing strategy and flow-control method heavily relies over the network topology. Hence, selecting the preferable network is the first choice for the MPC system. On the other hand, network power is also affected by the router radix.

Interconnection networks allow the send and receive of packets in the MPC systems over the interconnected cores. Hence, the research on interconnection networks has been quite wide, ranging from simple bus network to complex heterogenous networks. Some of them are designed with the graph property only and never been implemented in real MPC systems. Moreover, interconnection networks are classified with the direct and indirect interconnect. Direct networks maintains point to point connectivity with different cores according to the network topology. However, indirect networks don't follow the point to point connectivity; rather it connects any two core through the some switches. Multistage interconnection networks are the classic example of the indirect networks [18]. In this dissertation, we have considered only the direct networks.

Direct network topologies are defined by the graph theories. Hence, each node in graph theory is presented as the single processing element in MPC systems. However, we refined the nodes as the core, which is the individual processing element resides in each processor. On the other hand, edges are referred as the link between the cores. Graph theory also ensures the symmetry or asymmetry nature of the graph. In case of networks, it can be symmetric if it confirms its isomorphism to itself with any core. Symmetric network ensures substantial advantages over the asymmetric networks. The most important feature is the network scalability. Symmetric networks allow to expand in a modular fashion. Then, it ensures the use of simple routing algorithm, which is also effective for the message passing parallel programs.

Table 2.1: Commercially used networks in MPC systems

Machine	Networks
Connection Machine CM-5 [8]	Fat-Tree
Intel iPSC-2 [9, 10]	Hypercube
Intel Paragon [15]	2DMesh
Standford DASH	2DMesh
MIT J-Machine [13, 14]	3DMesh
Michigan HARTS	Hexagonal Mesh
Cray T3D [11]	3DTorus
Cray T3E [12]	3DTorus
Tera Computer	Incomplete 3DTorus
K Computer [1]	5/6DTorus
Blue Gene/q [29]	5DTorus
Sunway TaihuLight[4]	2DMesh

The modern interconnection networks of the MPC systems mainly focus on the fixed router radix because it is very important to maintain fixed router cost with the increased scalability. Increased router radix has the performance efficiency but also increases the router cost and also the power usages for the increased link connectivity and router activity. On the other hand, a constant router radix network allows to construct very large networks from the lowest network module. This regularity and modularity ensures that direct networks are often considered for the MPC systems. Table 2.1 shows the several topologies that are considered for various MPC systems.

2.2 General Terminology

Interconnection network helps to interconnect the processors as well as the memory modules. This section defines the basic terminology for the interconnection network and we also follows those terminology throughout the thesis.

Interconnection Network: Interconnection network acts as a graph where each vertices are defined as single computational unit and edges are defined as the communication links.

Core: A core acts as an individual computational unit in a massively parallel computer system. It consists of computational unit, local memory, supporting devices and a router.

Communication Link: A communication link acts as a data transmission path for one core to another. Communication links can be unidirectional or bidirectional.

Buffers: Buffers helps to hold the data before the packet could be delivered to the network. Hence, every communication channel in each core requires certain number of buffers to hold the packet.

2.3 Related Works

Networks are the backbone of a MPC system. MPC system heavily depends on the interconnected channels for communicating with other CPU cores or its memory system. However, earlier research on the interconnection networks was considered only conventional structures, which caused the degradation of network performance as well as the increased off-chip power usage. Furthermore, power estimation in CMOS VLSI chip shows that off-chip driven power can be scaled up to 65% of the total power consumption [16]. In addition, Infiniband QDR 40Gbps switch requires typically about 1W of electrical power for per link. Hence, the power consumptions have been heavily affected with increased off-chip connections. On the other hand, the most important feature for hierarchical interconnection networks (HIN) is the ability to maintain variable link structure for different network levels, which leads to the reduced off-chip power. Reduction in the power usage of the MPC systems considers three possibilities- (1) millivolt switch, (2) memory in 3-D and (3) specialized network architecture. Hence, our consideration for reducing the power usage is based on the network architecture. 16-tile MIT RAW on-chip network requires 36% of the total chip power [46]. In addition, in an Alpha 21364 microprocessor, the integrated routers and links consume about 20% of the total chip power (about 25W of total chip power 125W) (a 128 core 2Dtorus network fabricated with 180nm fabrication process) [28]. To achieve the exa-scale performance, one of the most challenging problems for the modern supercomputers is the reduction of current power consumptions. In addition, the requirement of power usages can be even scaled up to more than 300MW (which is nearly equal to the one nuclear power plant) with the conventional networks [17]. On the other hand, high degree networks show much better performance than the low degree of networks. However, high degree networks require higher power usage for their high degree of interconnected links.

2.4 Conventional Networks with Mesh and Torus Interconnect

Mesh [19] is one of the k-ary n-cube networks, which is very well-known for field of interconnection network and easy to layout in on-chips as well as for the off-chip connectivity. 2DMesh has been used in Sunway supercomputer with the achieved performance of 93 petaFlops. Average latency of Mesh network is O(sqrt(N)) with O(N) cost. Mesh network requires lowest number of virtual channels for the deadlock-freeness. However, it can't ensure the suitable network communication performance with the large network size. Figure 2.1 shows the network structure for 2DMesh network.

Torus also resides as another k-ary n-cube network [20]. However, Torus network considers extra wraparound links over the Mesh network. Hence, Torus requires 2 VC for its deadlock-free. Modern supercomputer like- Blue Gene/Q considers 5DTorus network as the interconnecting module with the achieved performance of 17 petaFlops performance included with the 18 core on-chip. Figure 2.2 shows the network architecture for 3DTorus.

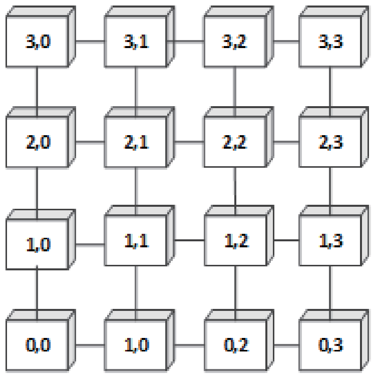


Figure 2.1: Network structure of 2DMesh Network

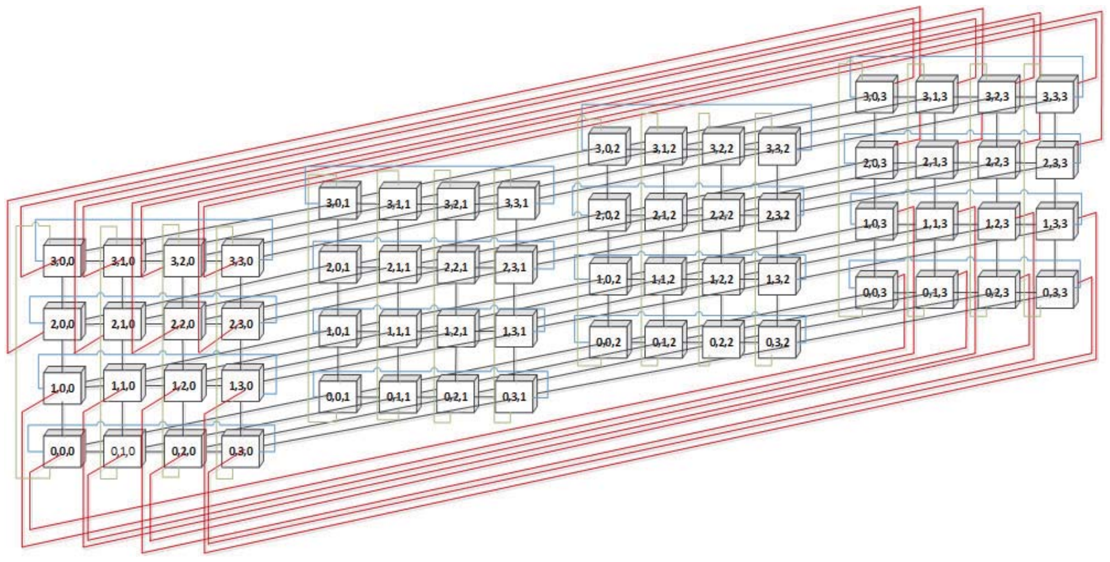
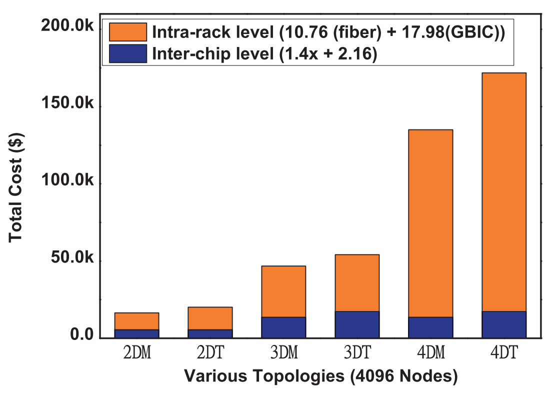


Figure 2.2: Architecture of 3DTorus Network

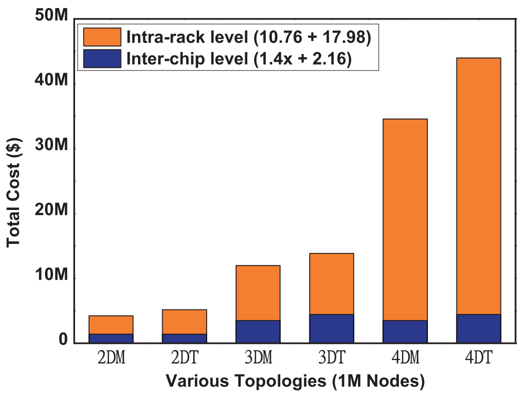
Table 2.2: Total number off-chip interconnect with 4K Cores

	# Inter-chip links	# Intra-rack links
2DMesh	1536	384
2DTorus	1536	512
3DMesh	3840	1152
3DTorus	4864	1280
4DMesh	3840	4224
4DTorus	4864	5376

System cost of MPC systems is highly correlated with increased number of wiring interconnect. High degree conventional networks require high number of wiring complexity, which increases the system performance and also increases the system cost. For example, the Modern super-computers like- Blue Gene/Q considered high degree network (5DTorus) for its own interconnect. However, the latest top ranked supercomputer Sunway Taihulight with 93 petaFlops performance considered the 2DMesh network to reduce the total system cost. Figure 2.3 shows system cost with respect to chip-chip links (level-2 links) and intra-rack links (level-3 links) (Figure 2.3(a) is the cost for the 4096 cores and Figure 2.3(b) is the scaled cost for 1M cores). We considered the electrical links at the inter-chip level and optical links at the intra-rack level. The consideration for number of links are shown at Table 2.2. This analysis explains that 2DMesh network will require about 90.39% less amount of cost for designing level-2 and level-3 off-chip links than the 4DTorus network. On the other hand, figure 2.4 shows the power usages at the inter-chip and intra-rack links.

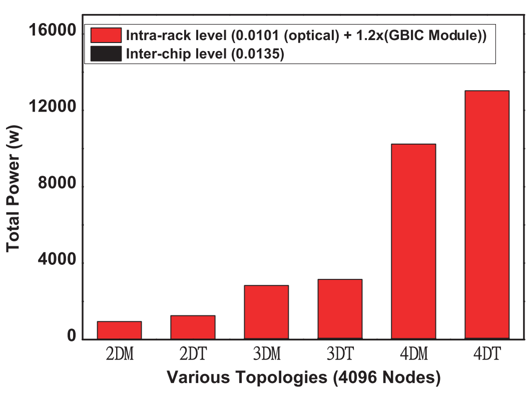


(a) Cost Analysis with 4K Cores

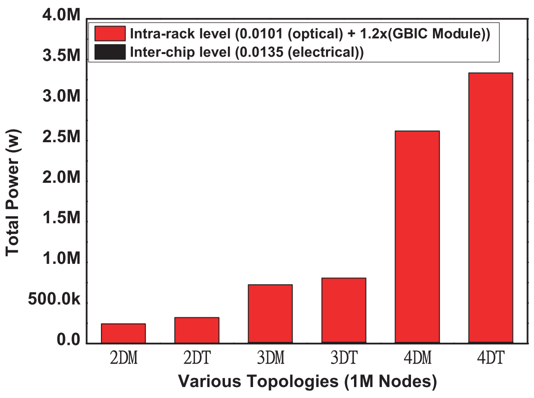


(b) Cost Analysis with 1M Cores

Figure 2.3: Wiring cost analysis on the conventional networks



(a) Power Analysis with 4K Cores



(b) Power Analysis with 1M Cores

Figure 2.4: Static Power analysis on the conventional networks

2.5 Hierarchical Interconnection Networks (HIN)

2.5.1 Tori-connected mESH (TESH) Network

The Tori-connected mESH (TESH) [21] network was introduced by V.K. Jain, et al.,is a hierarchical interconnection network. TESH consists of multiple basic modules (BM) that are hierarchically interconnected for building large scale system. The BM size of the TESH network is defined as $(2^m \times 2^m)$, where BM is the Level-1 network. On the other hand, higher level networks are interconnected as 2DTorus fashion. A TESH(m,L,q) can have number of cores at each level as $N = 2^{2mL}$; where m is any positive integer, L is network level and q is node interconnectivity. if m = 2, the size of the BM is (4×4). A BM of TESH (4×4) is shown in Figure 2.5.

2.5.2 Tori-connected Torus Network (TTN)

Tori connected Torus Network (TTN) [23, 24] was introduced by M.M. Hafizur Rahman, et al., considers the Torus connection over the Mesh connection. In the level-1 network or basic module level TTN considered the 2DTorus network and even at the off-chip level, it also considered the 2DTorus network. A $(2^m \times 2^m)$ on-chip module consists of a 2DTorus network of 2^{2m} processing elements (PE) and m is a positive integer. Figure 2.6 shows the on-chip module for TTN with m = 2, which defines the size of BM as (4×4). TTN requires 4 virtual channel for its deadlock-free routing.

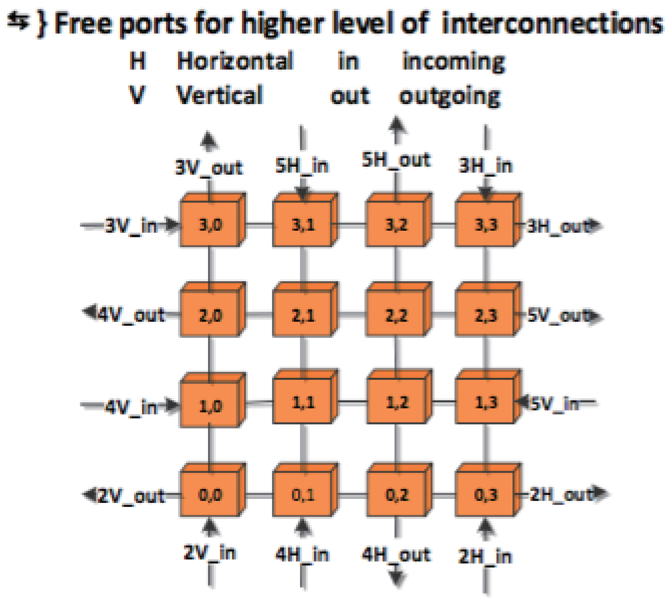


Figure 2.5: Basic Module of TESH Network [21]

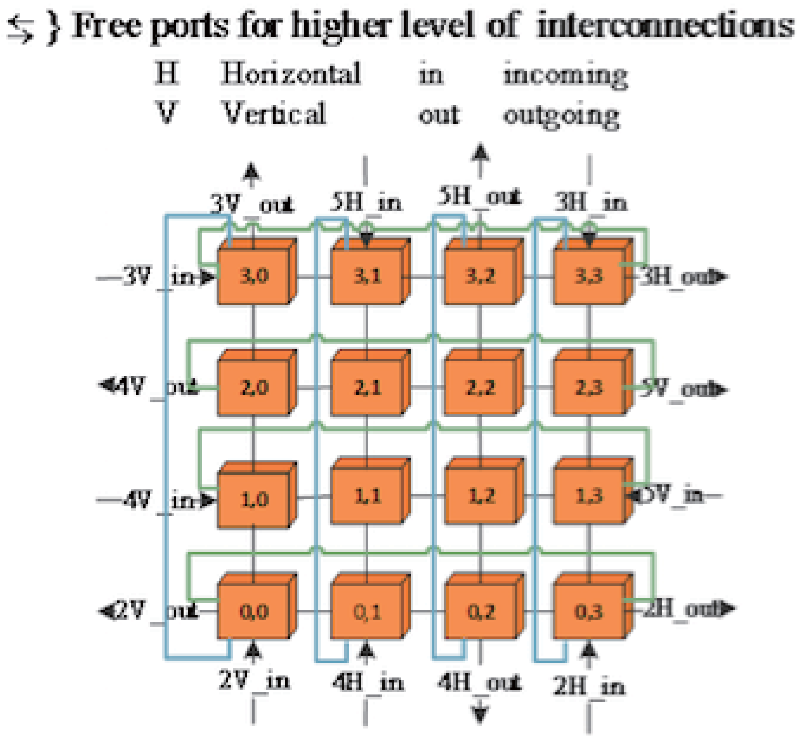


Figure 2.6: Basic Module of TTN Network [23]

2.5.3 Rectangular Twisted Torus Meshes (RTTM)

Rectangular Twisted Torus Meshes (RTTM) are similar to TESH network, which was proposed in 2010 [22]. This network also considers reducing the network diameter through the Mesh network at the on-chip level with ($2^m \times 2^m$) cores. However, the interconnection at the off-chip level is quite different than the TESH network, considers recursively interconnected ax2a next lower level sub-networks in the form of a rectangular twisted torus. This network has also high scalability with up to a millions of processors. Figure 2.7 shows the network interconnection for RTTM. RTTM ensures much better performance than the TESH network and also have the features of smaller diameter, shorter average distance and higher average link utilisation.

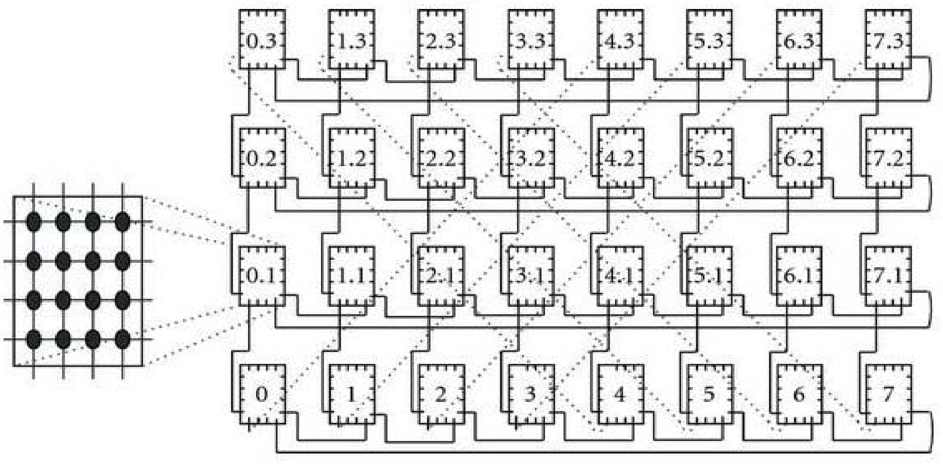


Figure 2.7: Architecture of RTTM Network [22]

2.5.4 Midimew Connected Mesh Network (MMN)

Midimew connected Mesh Network (MMN) considered the 2DMesh networks as the lowest level network, similar to TESH network, which was proposed in 2014 [25]. This network also considers reducing the network diameter through the Mesh network at the on-chip level with ($2^m \times 2^m$) cores. However, the interconnection at the off-chip level is midimew, considers different node connectivity for the off-chip level than the TESH network and RTTM network. This network has also high scalability with up to a million of processors. Figure 2.8 shows the basic module for MMN. A MMN(m,L,q) is also a hierarchical interconnection networks, where m is a positive integer, L is used for the network level and q is used for the inter-level connectivity. Upper level networks are connected with 2^{2m} immediate lower level of sub-networks considering ($2^m \times 2^m$) MInimal DIstance MEsh with Wraparound links (midimew) network. Figure 2.9 shows the level-2 network connectivity for the MMN.

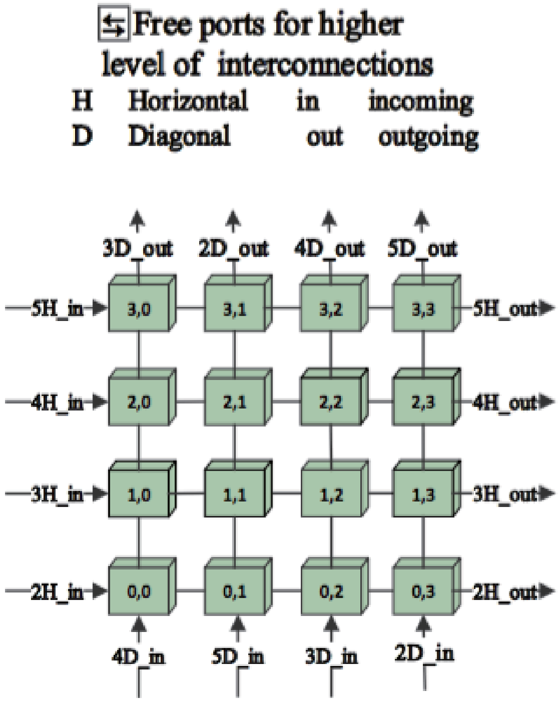


Figure 2.8: Basic Module of MMN

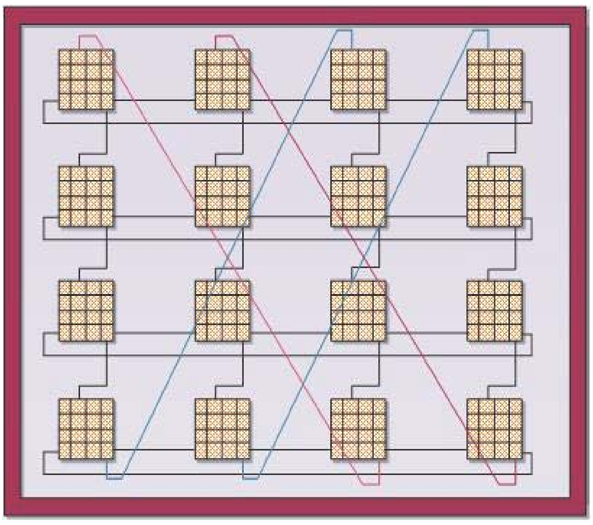


Figure 2.9: Level-2 Network Connectivity for MMN

2.6 Tofu Network

Tofu is a highly scalable network consists of 6D mesh/torus connectivity. Tofu(X,Y,Z,A,B,C) network considers XYZ connection for the intra-rack level and/or for the inter-rack connectivity. On the other hand, Tofu(X,Y,Z,A,B,C) considers ABC connection for the intra-chip level. Tofu network consists of 10 out going link. Hence, the node degree for Tofu(X,Y,Z,A,B,C) is 10. Each adjacent pair of ABC mesh/torus is interconnected with 12 links. Tofu considers extended dimension order routing and also requires 2 VC for deadlock-freeness [63]. Figure 2.10 shows the Tofu(X,Y,Z,A,B,C) network level interconnectivity. The on-chip module for Tofu consists of 12 cores. In this thesis (as comparing the network models of node level and the rack level), we have considered the 240 cores for the node level (similar to 256 cores of other networks), in total of 4080 cores at the each rack level for Tofu network. We have considered the Tofu(40,32,68,3,2,2) network with 1,044,480 cores as the 1M cores energy evaluation and the Tofu(20,16,17,3,2,2) network with 65,280 cores as the 65K cores energy evaluation.

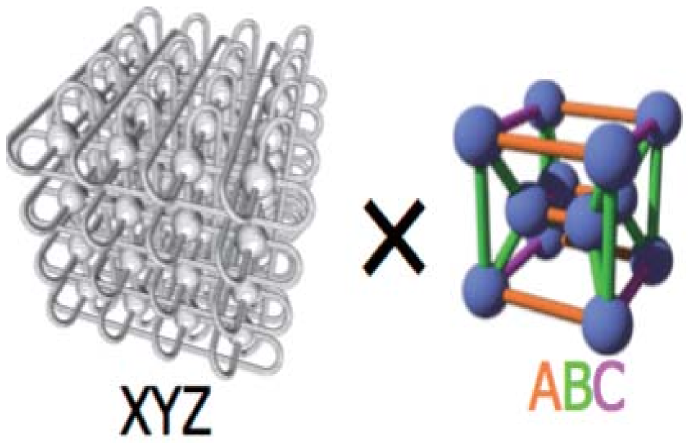


Figure 2.10: Network Connectivity for Tofu [1]

2.7 Requirements for Exa-scale Computing

Exa-scale computing related to the 'ExaFlops system'. One exaFlops (EFLOPS) system will be capable of performing about one quintillion (10^{18}) floating-point operations per second. This computational power is so large that any single human will require to perform one calculation in every second for 31,688,765,000 years to be the equivalence of a single second computational power of exa-scale computer and in the next generation computing requires this massive scale to improve the scientific fields (such as- 'Reducing Pollution'- can be possible through exa-scale computing, it will be possible to redesign the combustion systems in engines and increase the efficiency by 25-50%, 'Weather Prediction'- modern supercomputers are still lagging behind to make the proper weather forecasting due to the lack of computer performance, 'New Energy Solutions'- redesigning of wind turbines along with the solar panels will be an essential choice for the next decade). Modern supercomputers like- Sunway TaihuLight has already achieved about 93 petaFlops (PFLOPS) performance with 10,649,600 CPU cores. This constitutes that massive number of core scalability is required to meet this excessive demand. Most of the modern networks have poor scalability and even considers high router radix for meeting the high performance demand. Moreover, the high router radix increases the system cost as well as the power usage for the communicational links and routers (due to increase of processing activity in the local buffers) [36, 37].

2.8 Summary

This chapter we have introduced the general terminology for this thesis. The area of interconnection networks is quite wide. However, energy efficient supercomputing is a very new scope for this research. This chapter defines some useful comparators for energy efficient computing and we would maintain those comparator networks throughout the thesis. This chapter also explains the requirement for exa-scale supercomputing. Moreover, it also explains the reasons of avoiding the high degree networks for using in the large scale system.

Chapter 3

Network Design of HFBN

3.1 Introduction

The interconnection network plays the role to interconnect the multiprocessors and its memory module [1]. Hence, the performance of high performance computing (HPC) systems are likely depends on the network design. Since every computer chip has limited processing power, sequential processors cant be the suitable choice. However, the requirement for exa-scale computing will require about 13 million of connecting such processors. Hence, the requirement of interconnecting network has a huge impact for building exa-scale systems. Furthermore, todays one of the most powerful supercomputer Sunway Taihulight System has already achieved about 93 petaFlops performance with 10,649,600 cores requiring about 15.3MW electrical power using 2D Mesh interconnect [4]. The main concern for exa-scale systems are the reduction of power usage; for example- Sunway system will require about 168.3MW electrical power in achieving the exa-scale system with the modern advancements [4]. On the other hand, the network with high saturation rate will be obviously a poor consideration for the exa-scale system along with the network power usage is also another big concern as our target to build an exa-scale system within 20MW [27]. For example, in Alpha 21364 microprocessor, the network routers and links consume about 20% of the total chip power usage (25W of total chip power 125W) using 128 core 2DTorus network fabricated with 180nm process [28]. Furthermore, Off-chip links requires long interconnecting distance, causing much higher off-chip power than on-chip.

3.2 Theoretical Design of HFBN

3.2.1 Physical Aspects of Different Network Layers

Modern supercomputers are designed with various levels of hierarchy. For example- on-chip level as the lowest level network and off-chip level as the upper level network. Hence, the number of cores have the limited numbers at the on-chip level. For example- Blue Gene/Q chip was designed with 18 cores 64-bit A2 processor and 16 of those processor cores are used for computing with 5DTorus network having 2GB/s chip-to-chip links

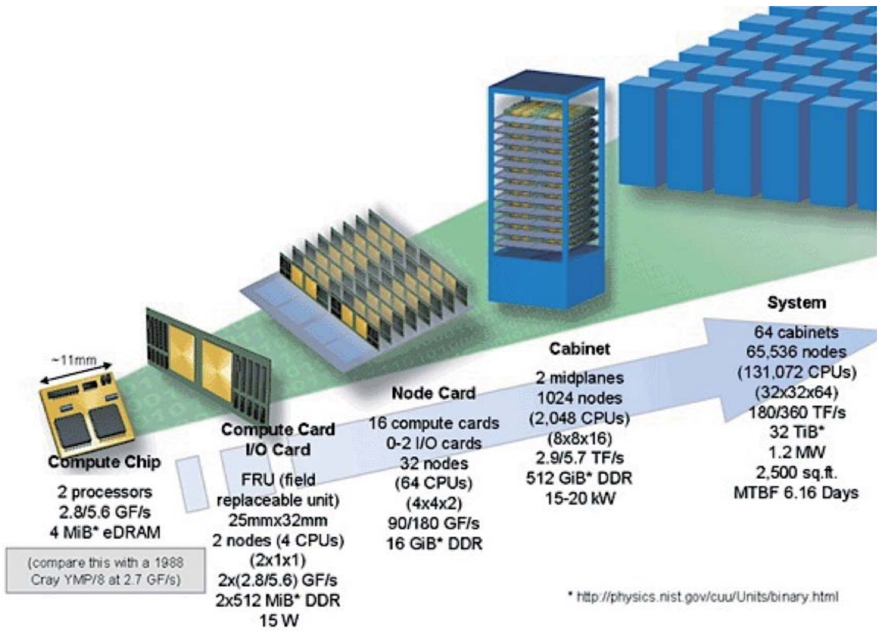


Figure 3.1: Block diagram for Blue Gene/L supercomputer [29]

[29]. However, those link cost is cheap, requires about pico-J power and aggregated bandwidth can be about 10GB/s. Figure 3.1 shows the design consideration for Blue Gene/L generation. Now, if we consider the network at an off-chip level, where next upper-level over the on-chip level is the node level. Node level considers multiple on-chip module to be interconnected, which can be connected through the Peripheral Component Interconnect Express (PCI express) [29]. Hence, the link cost is little higher than the chip level, requires nano-J power and bandwidth is about 1GB/s-10GB/s. Considering about the off-node connections, which are basically designed with the cabinet switches. And certainly, the cost will also increase, requires the micro-J power, and bandwidth can be up to GB/s speed [30]. Finally, the rack to rack connection is the most expensive in cost, requires about 10 micro-J power and have large latency. In modern supercomputers Infiniband switches are often considered at the rack to rack level. The physical layer of Infiniband has the bidirectional links of 2.5Gb/s for single data rate (SDR). However, for quad data rate (QDR) signalling permit single links to be scaled up to 10Gb/s (1.25GB/s $>$ 1GB/s) [31]. However, the per link power usage of 4x 10G QDR duplex transmission is typically about 1W for InfiniBand and per link cost is about \$300 [32].

3.2.2 Design Consideration and Analysis for On-chip and Off-chip Network

Hierarchical networks are the most desired networks for the next generation exa-scale supercomputers as the energy usage can be highly improved with comparing over the conventional networks. And, the physical aspects of modern supercomputers requires to have the hierarchical architecture at the various layers of networks. This is the main

Table 3.1: Evaluated static performance for various networks

64 Node Evaluation	2DMesh	2DTorus	FB	1DRING	3DMesh	3DTorus
Diameter	14	8	2	32	9	6
Average Distance	7	4	1.77	16	4	3

Table 3.2: Simulation Condition for On-chip Energy consumption

Parameter	Value	Units
number of cores	64	-
Fabrication process	65nm	-
Link length	3	[mm]
Operating freq.	1×10^9	Hz
Transistor type	HVT	-
Supply voltage	1	V
Traffic pattern	Uniform Traffic	-
Packet Size	6	flits
Simulation Cycle	5,000	-
Virtual Channels	2	-
Buffer Size	4	-
VA Model	VC_select	-
VA Buffer Model	SRAM	-

consideration of ours to have a heterogeneous network design through the various layers of network. In our on-chip network, we have considered the high performance network and off-chip level is considered with the low powered networks. Hence, we have evaluated the static performance of various networks with diameter and average distance. Table 3.1 shows this analysis and it clearly confirms the 2D Flattened Butterfly (FB) has the superiority than any other networks. As this research focus on the efficient energy usage, we also focus on the energy consumption at the on-chip level to consider the suitable network as the on-chip level. To evaluate the on-chip energy consumption, we considered the Orion energy model [33] along with the garnet 1.0 network simulator [34]. This evaluation considers only 64 cores. Figure 3.2 shows this analysis and it confirms that 2D Flattened Butterfly (FB) ensure much better energy usage over the 2D and 3D networks. On the other hand, Figure 3.3 shows the network energy usage for various high degree networks with 4K analysis. We considered the electrical links at the inter-chip level and optical links at the intra-rack level based on the evaluation of Section 2.4. This analysis is considered with the static performance parameter as the average distance and the total link power considering a single rack module. This analysis explains that 2DTorus network can ensure the low network energy usage over the other high degree networks and also shows slightly better than 2DMesh network. Hence, we would like to use "2D Flattened Butterfly" as the on-chip network (with fixed router radix) and the 2DTorus network as the off-chip interconnect for our hierarchical interconnect.

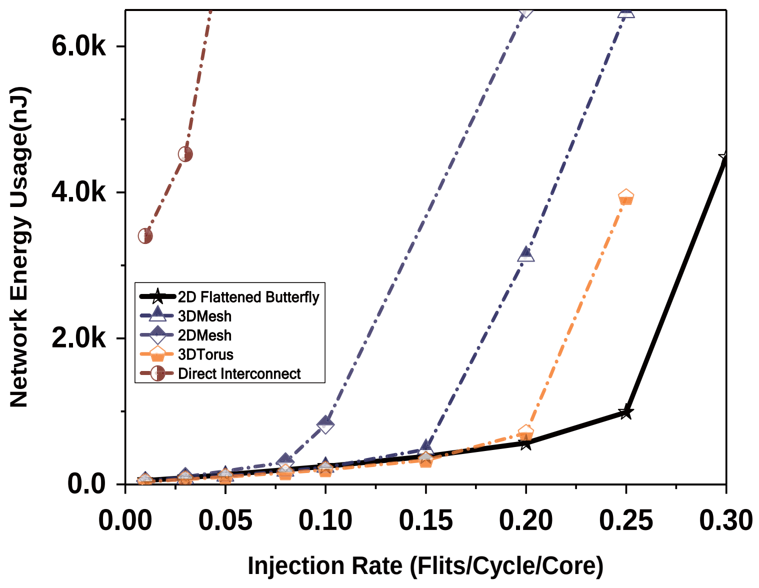


Figure 3.2: On-chip Energy Consumption

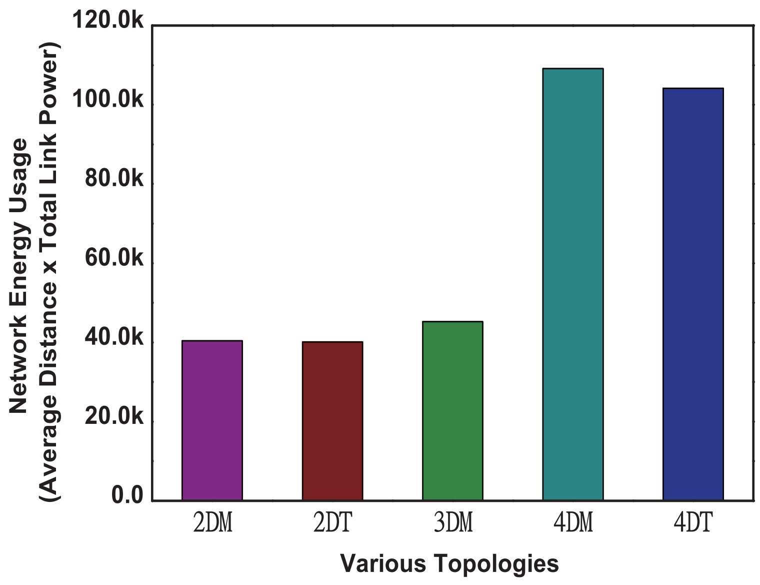


Figure 3.3: Off-chip Energy Usage (static analysis)

3.2.3 Definition of Hierarchical Flattened Butterfly Network (HFBN)

HFBN is considered as a hierarchical network as it maintains different topological pattern at the different levels of network structure. The lowest network level is (level-1 network) is defined as the basic module for HFBN, where each core maintains fixed number of radix and similar to 2D flattened butterfly architecture [35, 37]. On the other hand, the upper level of HFBN is considered with 2DTorus network. Hence, we named our network as the "Hierarchical Flattened Butterfly Network (HFBN)". This section defines the architectural pattern for HFBN, on-chip connections as well as for off-chips. HFBN maintains particular higher-level link pattern along with the 2DTorus upper level connectivity. On the other hand, the requirements for exa-scale system can be possible through interconnecting hundreds of millions of cores, which is certainly be possible by HFBN. However, HFBN requires pre-defined port assignments for its upper level connectivity. Figure 3.4 illustrates the interconnection philosophy of HFBN. However, we defined the HFBN through the definition network structure at various network levels and the equations to obtain fixed structure of HFBN as below-

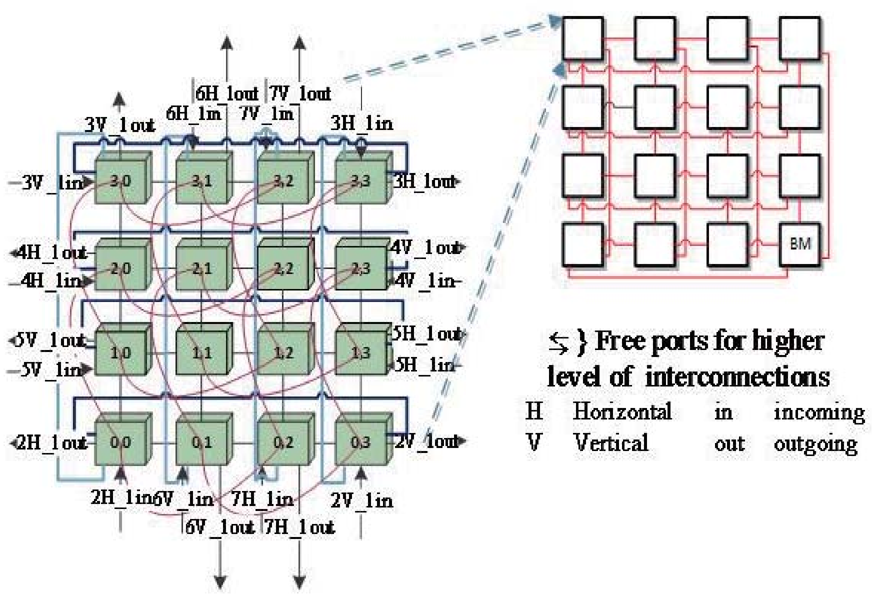


Figure 3.4: Interconnection of HFBN

Topological Definition: A HFBN(m, L, q) network, by definition is built with constant radix (similar to 2D flattened butterfly network) at the lowest level of the network followed by the 2DTorus interconnection at the upper level (with the particular connectivity consideration); where L considered as the level of hierarchy, q is the number of paired connectivity for each higher levels and m is any positive integer, which indicate the size of the basic module.

A HFBN(m, L, q) follows the exact definition for its certain level of connectivity-

Definition of HFBN Basic Module (BM): $(2^m \times 2^m)$ is the lowest network level, (m = is any positive integer)

Definition of HFBN Upper-level Connection: $L_{max} = \lceil 2 \times (2^m - 1)/q \rceil + 1$ is the maximum network size. Q_{max} = the maximum possible paired connectivity for any value of m; $Q_{max} = 2(2^m - 1)$. $1 \leq$ q $\leq Q_{max}$; depending on the value of q, HFBN(m, L, q) considers two set of configuration-

(a) IDEAL_HFBN(m, L, q) when $(2 \times (2^m - 1) mod\ q) = 0$

(b) PARTIAL_HFBN(m, L, q) when $(2 \times (2^m - 1) mod\ q)\ ! = 0$ (some exterior cores will be remained free).

Link Connectivity:

BM cores requires two digit for the formulation; the first is the Y-index, then the X-index. In general, in a Level-L HFBN, the core address can be represented by:

$$A^L = \{\ (a_{yL},\ a_{xL}) \qquad \text{if } L_{max} \geq L \geq 1$$

More generally in a Level-L HFBN, the core address is represented by-

$$\begin{aligned} A &= A^L\ A^{L-1}\ A^{L-2}\ ...\ ...\ ...\ A^2\ A^1 \\ &= \ (a_{2L-1},\ a_{2L-2})\ (a_{2L-3},\ a_{2L-4})...\ ...\ (a_3,\ a_2)\ (a_1,\ a_0) \end{aligned} \tag{3.1}$$

Here, the Level-1 is defined by core address (a_1, a_0), where a_1 defines the core address for the Y-axis and then the X-axis with the a_0. Higher level networks are two dimensional networks, hence we consider the first digit as the row index and then the second one is the column index. Now, if the address of a core N^1 included in BM_1 is represented as $N^1 = [(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ s_0)]$ and the address of a core N^2 included in BM_2 is represented as $n^2 = [(d_{2L-1},\ d_{2L-2})...\ ...(d_3,\ d_2)\ (d_1,\ d_0)]$. The core N^1 in BM_1 and N^2 in BM_2 are connected if the following connections are satisfied for N^2-

- **Link for BM-**

  $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ s_0)]$ to $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ s_0 \pm 1 \mod 2^m)]$ where $2^m > s_0 >= 0$,

  $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ s_0)]$ to $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1 \pm 1 \mod 2^m,\ s_0)]$ where $2^m > s_1 \geq 0$

  $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ s_0)]$ to $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ ((s_1 + 2^m/2) \mod 2^m,\ s_0)]$

  $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ s_0)]$ to $[(s_{2L-1},\ s_{2L-2})...\ ...(s_3,\ s_2)\ (s_1,\ (s_0 + 2^m/2) \mod 2^m\ )]$

- **Link for Higher Level Vertical Connections-**

  Connected Core, $D = (((BMN^{L-1} \times 2^m) \pm x) \mod BMN^L)$

- **Link for Higher Level Horizontal Connections-**

  Connected Core, $D = (((BMN^{L-1}) \pm x) \mod (BMN^{L-1} \times 2^m))$

In case of higher level links, BMN ($BMN = (2^m \times 2^m)$) defines the number of cores in a basic module and L defines the core number of corresponding levels. For example, if m is 2, then the BMN = 16. On the other hand, x defines the source core number which is equal to $(s_{2L-1} \times 2^m,\ s_{2L-2})\ \times BMN^{L-1}$ +... ...+$(s_3 \times 2^m + s_2)\ \times BMN$ + $(s_1 \times 2^m + s_0)$. The highest level of network, which can be obtained by a $(2^m \times 2^m)$ BM is defined by $L_{max} = \lceil 2 \times (2^m - 1)/q \rceil + 1$. Algorithm 1 shows the port assignment for upper-level connectivity for a particular basic module. And, the flowchart of this algorithm is given in Figure 3.5. This algorithm considers all of the exterior cores in the on-chip network to be interconnected with the other on-chip module. Algorithm 1 requires the input value of m and q. On the other hand, L_{max} can be calculated from m and q. Function HIGHERLevel_HFBN(m,q) allocates the particular core in the basic module for high level port connectivity, which requires the value of m, possible number of paired connectivity for each level. As the high level ports are being allocated by the exterior cores of the each basic module, this algorithm allocates the each possible port position for the higher level connectivity. In the initialize function, L_{max} has been calculated and BMAX is the number of possible cores in each X or Y directions.

Algorithm 1 Higher Level port assignment for HFBN

```
HIGHERLevel_HFBN(m,q); /* Main function to call with m, possible number of paired connectivity for each level */
    initialize(q); /* Initializing the pre-defined ports */
    for lv = 2 : $L_{max}$; /* lv defines the network level */
      for $i = 1$ : q; /* loop for making sure about all paired connectivity */
        $min\_dist$ = 65536;
        if (lv ! = 2 and $i$ = 1)
          lvmh = $port\_levelQ(lv-1,'H',q)$; /* assign the previous level Horizontal port address */
          lvmv = $port\_levelQ(lv-1,'V',q)$; /* assign the previous level Vertical port address */
        else
          lvmh = $port\_levelQ(lv-1,'H',i-1)$; /* assign the previous level Horizontal port address */
          lvmv = $port\_levelQ(lv-1,'V',i-1)$; /* assign the previous level Vertical port address */
        endif;
        if (!availableQ(lv, 'V', i)) /* check the current level port is already assigned or not */
          for (x=0, y = 0; x $\leq$ BMAX; x++, y++)
            SaveXY( x, y, &$save\_pv$, &$save\_ph$, lvmv, lvmh, dist, &$min\_dist$); /* this function returns the y and x port position of current level through $save\_ph$ and $save\_pv$ */
          endfor;
          setHVLQ( $save\_ph$.y, $save\_ph$.x, lv, 'H', i ); /* initialize the port */
          setHVLQ( $save\_pv$.y, $save\_pv$.x, lv, 'V', i); /* initialize the port */
        endif;
      endfor;
    endfor;
end
SaveXY(x,y,*$save\_pv$,*$save\_ph$, lvmv, lvmh, dist, *$min\_dist$); /* this function returns the y and x port position of current level through $save\_ph$ and $save\_pv$ */
  if( LV[0][x] = 0 )
    pv = $set\_xy(0,x); ph = set\_xy(BMAX,x)$; /* consider the Vertical core position with respect to x */
    dist = distance(pv, lvmh) + distance(ph, lvmv); /* compare the distance from the previous level outgoing ports */
    if( dist < $min\_dist$ )
      $save\_pv$ = pv; $save\_ph$ = ph; $min\_dist$ = dist; /* save the position */
    endif;
    ph = $set\_xy(0,x); pv = set\_xy(BMAX,x)$; /* consider the horizontal core position with respect to x */
    dist = distance(pv, lvmh) + distance(ph, lvmv); /* compare the distance from the previous level outgoing ports */
    if( dist < $min\_dist$ )
      $save\_pv$ = pv; $save\_ph$ = ph; $min\_dist$ = dist; /* save the position */
    endif;
  endif;
  if( LV[y][0] = 0 )
    pv = $set\_xy(y,0); ph = set\_xy(y,BMAX)$; /* consider the Vertical core position with respect to y */
    dist = distance(pv, lvmh) + distance(ph, lvmv); /* compare the distance from the previous level outgoing ports */
    if( dist < $min\_dist$ )
      $save\_pv$ = pv; $save\_ph$ = ph; $min\_dist$ = dist; /* save the position */
    endif;
    ph = $set\_xy(y,0); pv = set\_xy(y,BMAX)$; /* consider the horizontal core position with respect to y */
    dist = distance(pv, lvmh) + distance(ph, lvmv);
    if( dist < $min\_dist$ )
      $save\_pv$ = pv; $save\_ph$ = ph; $min\_dist$ = dist; /* save the position */
    endif;
  endif;
end;
```

```
typedef struct /* defining the core struct with x, y position */
    position x, y;
end;
distance( src, dest ) /* compare the distance from the previous level outgoing ports with the current possible location, src is the current port position and dest is the previous assigned level port position */
    dist.x = abs( src.x - dest.x );    dist.y = abs( src.y - dest.y );    return( dist.x + dist.y );
end;
availableQ( pl, phv , connect) /* check the current level port is already assigned or not (pl is the level number and phv is for the choice of vertical or Horizontal selection and connect is to defined the current core number) */
    for( p.y=0; p.y $\leq$ BMAX; p.y++ )
      for( p.x=0; p.x $\leq$ BMAX; p.x++ )
        if( pl == LV[p.y][p.x] && phv == HV[p.y][p.x] && connect == IN[p.y] [p.x]) return true;
    return false;
end;
initialize(q) /* Initialising the pre-defined ports */
    BMAX = $2^m$ - 1;
    $L_{max}$ = ceil(2*($2^m$ - 1)/q) + 1;
    for( x=0; x <= BMAX; x++ )
      for( y=0; y <= BMAX; y++ )
        LV[y][x] = 0; HV[y][x] = '*'; IN[y][x] = 0; /* LV stands for level number and HV is for the define the vertical or horizontal out-going
    if (q < 2)
      setHVLQ(0, 0, 2, 'H', 1); setHVLQ( 0, BMAX, 2, 'V',1 ); setHVLQ( BMAX, 0, 3, 'V',1 ); setHVLQ(BMAX, BMAX, 3, 'H',1 );
    else
      setHVLQ(0, 0, 2, 'H', 1); setHVLQ(0, BMAX, 2, 'V', 1);
      setHVLQ(BMAX, 0, 2, 'V', 2); setHVLQ(BMAX, BMAX, 2, 'H', 2);
    endif;
end;
port_levelQ( pl, phv , connect) /* check the previous level which is already assigned (pl is the level number and phv is for the choice of vertical or Horizontal selection and connect is to defined the current core number) */
    for( p.y=0; p.y $\leq$ BMAX; p.y++ )
      for( p.x=0; p.x $\leq$ BMAX; p.x++ )
        if( pl == LV[p.y][p.x] && phv == HV[p.y][p.x] && connect == IN[p.y] [p.x]) return( p );
end
setHVLQ(y, x, pl, phv, connect) /* initialize the LV, HV and IN array */
    LV[y][x] = pl;
    HV[y][x] = phv;
    IN[y][x] = connect;
end
set_xy(y, x ) /* returns the core structure */
    p.x = x; p.y = y;
    return(p);
end
```

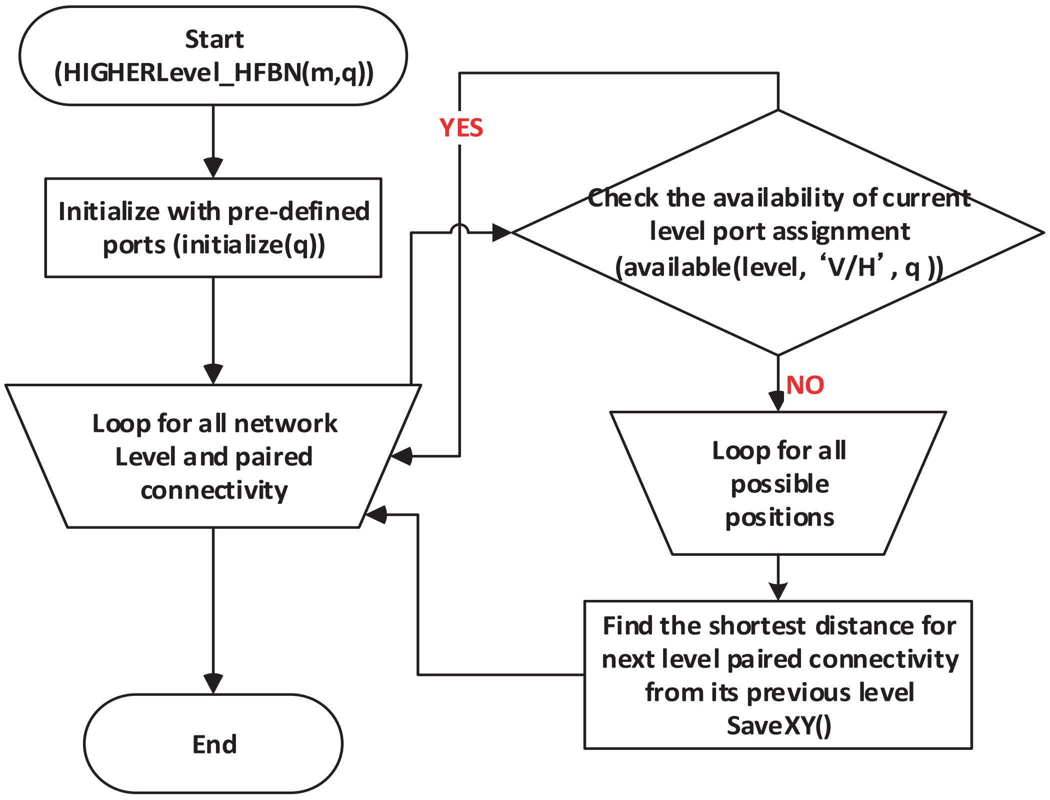

Start (HIGHERLevel_HFBN(m,q))
Initialize with pre-defined ports (initialize(q))
Loop for all network Level and paired connectivity
End
YES
Check the availability of current level port assignment (available(level, 'V/H', q))
NO
Loop for all possible positions
Find the shortest distance for next level paired connectivity from its previous level SaveXY()

Figure 3.5: Flowchart for Port Assignment Algorithm

3.2.3.1 Discussion of Size of the HFBN Basic Module(BM)

HFBN(m, L, q) used only six intra-chip links for the interconnection of the basic module. Hence, the on-chip design for HFBN and flattened butterfly has a distinctive difference. However, the basic module design for HFBN(2, L, q) follows the same pattern as the flattened butterfly network. Since HFBN maintains a constant node degree, the HFBN link pattern varies (link connectivity has already been defined beforehand for the basic module) from flattened butterfly when m is greater than 2. The lowest level of HFBN(m, 1, q) has been considered as the "Basic Module" (BM). HFBN considers $(2^m \times 2^m)$ number of cores as his basic module size. Hence, m = 2 means the possible number of cores at the BM level will be sixteen. Figure 3.6 shows the basic module of HFBN(2, L, q). This network is based on the two dimensional architecture and hence, BM connectivity considered with X and Y directions. However, we have already discussed about the link connectivity for the basic module beforehand.

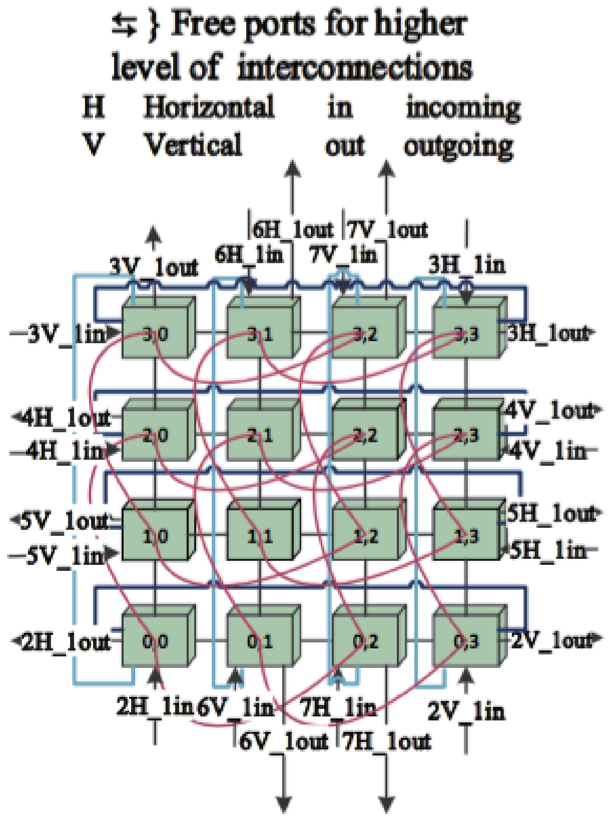


Figure 3.6: A (4 × 4) Basic Module of HFBN

3.2.3.2 Discussion of Size of the Upper-level HFBN

Integrating large number of on-chip links is useful for network performance as well as cost effective. However, the scenario for off-chip level is completely different, where per link cost and power requirement increases simultaneously with the total system cost. Hence, we have considered the hierarchical design for the next generation supercomputer architecture, where particular level links are interconnected for each level. Higher level of HFBN considers 2DTorus interconnection considering recursive interconnect patterns of the immediate lowest level of subnetworks. Hence, a level-2 (node level) HFBN consists of a certain number (2^{2m}) of level-1 networks. This statement constitutes that a HFBN(2, L, q) will have 16 level-1 network or basic module for the complete level-2 network. Figure 3.7 shows the formation of a single level-3 network through the combination of 16 Level-2 networks and 256 Level-1 networks of HFBN(2, L, q). On the other hand, we have considered multiple lemmas to increase the readability of the network setup.

Lemma 3.1 A ($2^m \times 2^m$) basic module of HFBN has $2^3 \times (2^m - 1)$ numbered of free ports for the higher-level interconnectivity.

One of important points to achieve high performance, HFBN must use all its free ports. Hence, the large number of paired connectivity is highly effective. q is defined as the number of paired connectivity for each higher level and Q_{max} is the max paired value for any m; Maximum paired value for any m is defined as the $Q_{max} = 2(2^m - 1)$ and the q is all the possible divisors of Q_{max}. For example- $Q_{max} = 2(2^m - 1) = 2(2^2 - 1) = 6$. In addition, the highest level network of HFBN can be defined as $L_{max} = ceil(2 \times (2^m - 1)/q) + 1$. Hence, HFBN(2, L, 1) can be constructed as the maximum seven network level $L_{max} = (2 \times (2^2 - 1)/1) + 1 = 7$. Number of paired connectivity, q is responsible for the increase of number of outgoing and incoming connection at the each off-chip level. Increased value of q, increases the number of in/out connections and decreases the maximum network level. Figure 3.8 shows the architectural design of HFBN(2, 3, 3) and HFBN(2, 2, 6). Those figures also show that the choice of off-chip connectivity of a particular core with the paired connectivity number (starting 1 to 3 for Figure 3.8(a) and 1 to 6 for Figure 3.8(b)).

Lemma 3.2 The total number of the cores at each level of HFBN can be defined as $N = 2^{2mL}$

HFBN maintains a fixed number of cores at the basic module level $(2^m \times 2^m)$ and builds the upper level with having (2^{2m}) immediate lower level of subnetworks, which finally constitutes the network size of particular number of interconnected cores. For example, a HFBN(2, 3, q) has a network size of 4096 cores. Table 3.3 generalises the architectural parameter for HFBN(m, L, q). Table 3.4 compares the various levels of HFBN for m = 2 with the different q values with the possible number of interconnected cores at the different level of networks.

Table 3.3: Generalisation of HFBN

Basic Module	Number of Paired Connectivity	Max Levels, L_{max}	Number of Cores
$(2^m \times 2^m)$	$1 \leq$ q $\leq Q_{max}$	$\lceil 2 \times (2^m - 1)/q \rceil + 1$	$N_L = 2^{2mL}$

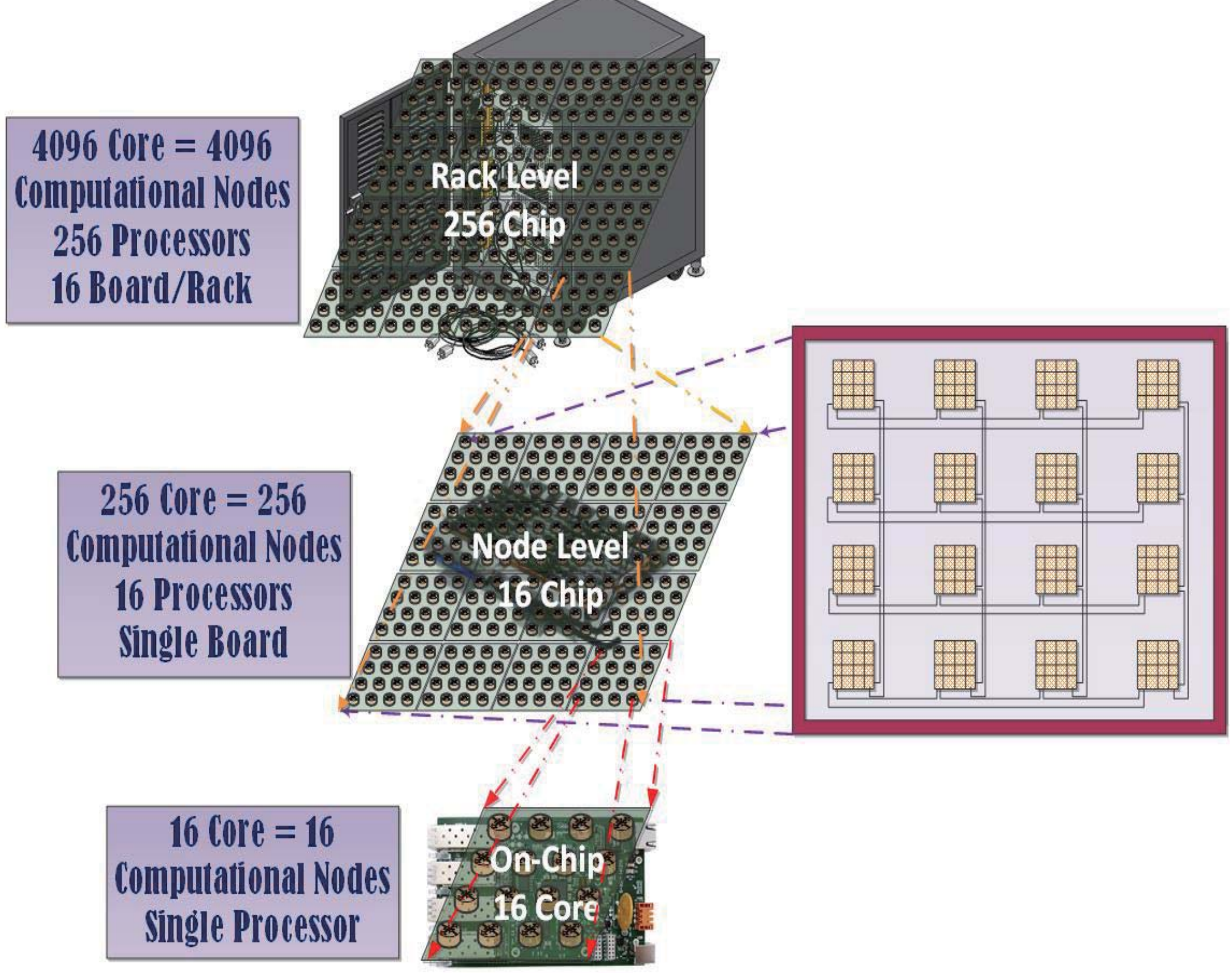
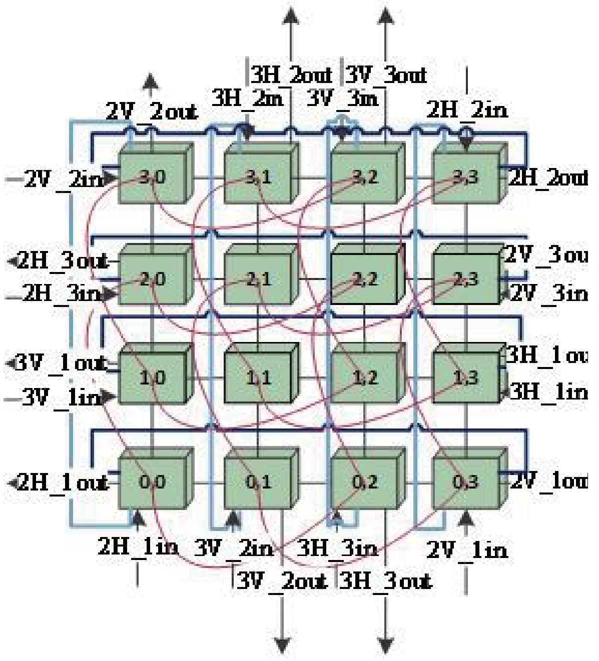


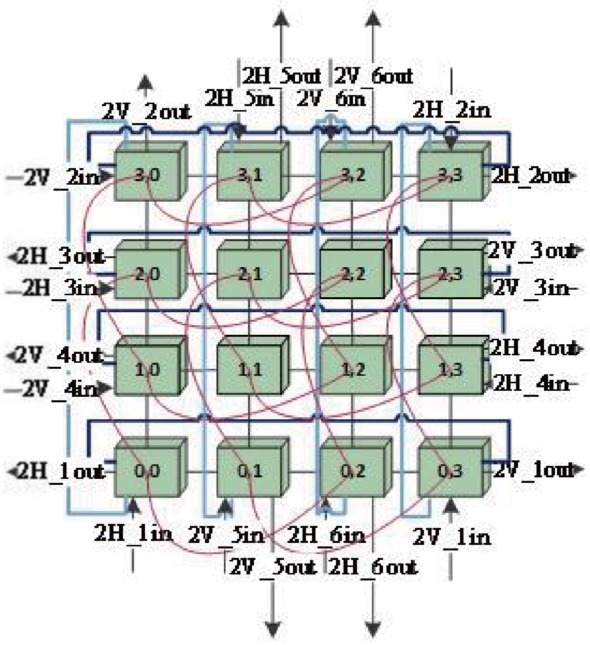
Figure 3.7: Higher-level of Interconnection for HFBN(2, L, q)

Table 3.4: Possible number of cores with various levels of HFBN

m	q	L	Number of Cores
2	1	3	$N_3 = (2^{2\times2\times3}) = 4{,}096$
2	2	3	$N_3 = (2^{2\times2\times3}) = 4{,}096$
2	1	4	$N_4 = (2^{2\times2\times4}) = 65{,}536$
2	2	4	$N_4 = (2^{2\times2\times4}) = 65{,}536$
2	1	5	$N_5 = (2^{2\times2\times5}) = 1{,}048{,}576$
2	1	6	$N_6 = (2^{2\times2\times6}) = 16{,}777{,}216$
2	1	7	$N_7 = (2^{2\times2\times7}) = 268{,}435{,}456$



(a) a 4 x 4 BM customised for HFBN(2,3,3)



(b) a 4 x 4 BM customised for HFBN(2,2,6)

Figure 3.8: Increased paired connectivity for each level

3.2.4 Number of Links at Various Network Levels

Interconnection network has one of the greatest concerns for its own router layout [30], [39]. The K-Computer requires cable length about 1,000 kilometers [30]. On the other hand, as per analysis on Infiniband QDR 40Gbps switch requires typically about 1W of electrical power for its per link [31]. Hence, the number of on-chip as well as the off-chip connection will be a major concern in designing the exa-scale system. Figure 3.7 shows the hierarchical structure for HFBN(2, L, q), where level-1 network constructs at the chip level, level-2 network will be used for node level and level-3 network can be used at the rack level. The number of interconnecting links at various layers of HFBN can be defined by Equation 3.2.

$$
\begin{aligned}
L_L = {} & (\text{Number of BM in current level, } L_L) \times \{\text{Number of Intra-links in} \\
& L_1 \text{ (48 links for m = 2) network}\} + \sum_{i=2}^{L} \{(\text{Number of BM at current level, } L_L) \\
& \times (\text{Number of exterior outgoing links at each higher-level, } L_i)\} \qquad [\text{where N} \geq 2]
\end{aligned}
\tag{3.2}
$$

Here, Equation 3.2 defines how to calculate the interconnected links for various levels of network. The level-1 HFBN(2, 1, 1) network requires about 48 links for its BM. We

Table 3.5: Generalisation of number of links at various levels of HFBN

Topology	Level-1 Network	Higher Level Network
HFBN	(Number of X-directional Links) + (Number of Y-directional Links)	(Number of BM in current level) $\times$ (Number of inner links in level-1 network) + $\sum_{i=2}^{L}$ { (Number of BM at current level, L_L) $\times$ (Number of outgoing links at each higher-level, L_i)} [where N $\geq$ 2]

Table 3.6: Example of required number of links at various levels of networks

Topology	Level-1 Network (16 Cores)	Level-2 Network (256 Cores)	Level-3 Network (4096 Cores)
2DMesh	24 links	480	16128
2DTorus	32 links	512	16384
HFBN(2,L,1)	48 links	16 x 48 (L1 Links) + 32 (L2 Links) = 800	13312

have also generalised this equation at Table 3.5. On the other hand, Table 3.8 shows the total number of links for HFBN, in which the required number of links for HFBN is above only 800 at the node layer. Table 3.6 also shows the link comparison on various networks like- 2DMesh, 2DTorus against the HFBN. And, from this table we can also find that 2DMesh and 2DTorus network require much higher number of links than the HFBN at the higher levels. HFBN(2, 3, 1) will require about 18.75% less interconnected links than the 2DTorus and about 17.46% less than 2DMesh network.

3.2.5 Routing Algorithm for HFBN

Deterministic routing has been the common routing consideration along with the adaptive routing for mitigating the network faults. Modern supercomputer *BlueGene/L* also uses deterministic routing along with the adaptive routing. Hence, in our performance analysis, we also considers a simple deterministic routing ($dimension - order$ routing (DOR)) for HFBN(2, L, 1) class. $Dimension - order$ routing continues to route the packet to the same dimension until the distance of that dimension become zero. Now considering HFBN routing, routing Algorithm 2 can be subdivided into two parts; one part considers the *BM_routing* and another part considers higher-level routing (*Routing_HFBN*). For example, if the packet is destined for the other BM, the source core will send the packet to the *outlet_core* of the next interconnected BM of the current network level. On the other hand, *receiving_core* is used to track down the new source core address after the BM transfer has been completed. If the packet is destined to another BM, the source core sends the packet to the outlet core which connects the BM to the level at which the routing is performed. Suppose, source core address is s = (s_{2L-1}, s_{2L-2}) (s_{2L-3}, s_{2L-4}) ... ... (s_3, s_2) (s_1, s_0) and destination core d = (d_{2L-1}, d_{2L-2}) (d_{2L-3}, d_{2L-4}) .. ... (d_3, d_2)

(d_1, d_0) considering the routing at the Y,X direction for the higher levels as well as for the level-1 networks. Similarly, routing tag can be defined as t = (t_{2L-1}, t_{2L-2}) (t_{2L-3}, t_{2L-4})... ...(t_3, t_2) (t_1, t_0), where t_i = d_i - s_i. In Routing_HFBN function, outlet_x and outlet_y are the function to get x coordinate s_1 and y coordinate s_2 of the core that link (s, d, l, dα) exists, where level l($2 \leq$ l $\leq$ L), dimension d(d $\in$ {V,H}) and direction α(α $\in$ {+,-}). Hence, vertical and horizontal direction are represented by V+, V-, H+ and H-.

As we considered the DOR routing for HFBN, Figure 3.9 shows the packet routing from the source core (1,2),(1,2),(1,2) to destination core (2,1),(2,1),(2,1). Hierarchical routing for HFBN send the highest network level at first. Hence, the packet will be routed to Level-3 network, the source core (1,2),(1,2),(1,2) will send the packet to the basic module outlet core (1,2),(1,2),(3,0) of Level-3 network and will reach Level-3(2,2) from Level-3(1,2) network. Similarly, after completing the level-2 routing, Level-1 routing will be started from (2,1),(2,1),(0,0) core and will finally reach the destination core.

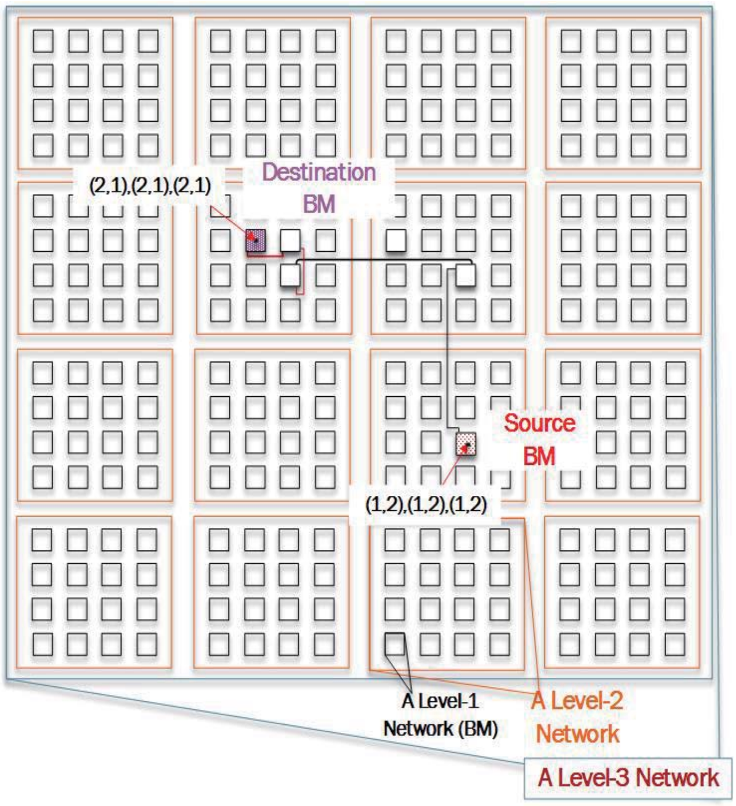


Figure 3.9: Routing Path for HFBN(2,3,1)

Algorithm 2 Routing Algorithm for HFBN(2, L, 1)

```
Routing_HFBN($s_{2L-1}$, $s_{2L-2}$, $s_{2L-3}$, , $s_1$, $s_0$, $d_{2L-1}$, $d_{2L-2}$, $d_{2L-3}$, , $d_1$, $d_0$);
tag: $t_{2L-1}$, $t_{2L-2}$, $t_{2L-3}$, , $t_1$, $t_0$;
    for i = 2L-1 : 3;
        if ((($d_i$ - $s_i$ + $2^m$) mod $2^m$) <= $2^m/2$) then routedir = positive; $t_i$ = (($d_i$ - $s_i$ + $2^m$) mod $2^m$);
        else routedir = negative; $t_i$ = ($2^m$ - ($d_i$ - $s_i$ + $2^m$) mod $2^m$); endif;
        while ($t_i$ ! = 0) do
            if (i mod 2) = 1, then $outlet\_core_x = outlet\_x(s, d, \lfloor (i)/2+1 \rfloor, H, routedir)$;
                                   $outlet\_core_y = outlet\_y(s, d, \lfloor (i)/2+1 \rfloor, H, routedir)$;
            else                   $outlet\_core_x = outlet\_x(s, d, \lfloor (i)/2+1 \rfloor, V, routedir)$;
                                   $outlet\_core_y = outlet\_y(s, d, \lfloor (i)/2+1 \rfloor, V, routedir)$; endif;
            $BM\_routing(s_1, s_0, outlet\_core_y, outlet\_core_x)$;
            if (routedir = positive) then send the packet to the next BM;
            else move the packet to previous BM; endif;
            if ($t_i$ > 0) then $t_i$ = $t_i$ - 1; endif;
            if ($t_i$ < 0) then $t_i$ = $t_i$ + 1; endif;
            if (i mod 2) = 1, $s_1 = receiving\_core_x(s, d, \lfloor (i)/2+1 \rfloor, H, routedir)$;
                              $s_2 = receiving\_core_y(s, d, \lfloor (i)/2+1 \rfloor, H, routedir)$;
            else              $s_1 = receiving\_core_x(s, d, \lfloor (i)/2+1 \rfloor, V, routedir)$;
                              $s_2 = receiving\_core_y(s, d, \lfloor (i)/2+1 \rfloor, V, routedir)$; endif;
        endwhile;
    endfor;
    $BM\_routing(s_1, s_0, d_1, d_0)$;
end
$BM\_routing(s_1, s_0, d_1, d_0)$;
source: $s_1, s_0$; destination: $d_1, d_0$;
$BM\_tag$: $t_1, t_0$ = destination address($d_1, d_0$) - source address($s_1, s_0$);
    for i = 0 : 1;
        if ($t_i$ > 0), movedir = positive; endif;
        if ($t_i$ < 0), movedir = negative; endif;
        if (movedir = positive and $t_i$ = $2^m/2$) then $t_i$ = $t_i$ - 1; endif;
        if (movedir = positive and $t_i$ = -$2^m/2$) then $t_i$ = $t_i$ + 1; endif;
        if (movedir = negative and $t_i$ = $2^m/2$) then $t_i$ = $t_i$ - 1; endif;
        if (movedir = negative and $t_i$ = -$2^m/2$) then $t_i$ = $t_i$ + 1; endif;
        if (movedir = positive and $t_i$ > $2^m/2$) then $t_i$ = $t_i$ - $2^m$; endif;
        if (movedir = negative and $t_i$ < -$2^m/2$) then $t_i$ = $t_i$ + $2^m$; endif;
    endfor;
    while($t_0$ ! = 0) do
        if ($t_0$ > 0) then move packet to +x core; $t_0$ = $t_0$ - 1; endif;
        if ($t_0$ < 0) then move packet to -x core; $t_0$ = $t_0$ + 1; endif;
    endwhile;
    while($t_1$ ! = 0) do
        if ($t_1$ > 0) then move packet to +y core; $t_1$ = $t_1$ - 1; endif;
        if ($t_1$ < 0) then move packet to -y core; $t_1$ = $t_1$ + 1; endif;
    endwhile;
end
```

3.2.6 Deadlock-free Routing for HFBN

Routing of packets requires to be deadlock-free, otherwise packet will not be sent to the destination core ever and will delay the delivery of other packets, which in turn, drastically reduces dynamic communication performance. In this section, we studied the deadlock-freedom for HFBN. HFBN(2, 1, 1) network maintains the core-core connections from each x or y (row/column) directional cores. Hence, there is no wrap-around routing is required for HFBN(2, 1, 1), which leads to a similar routing for Mesh network (which requires only one VC for routing). This conclusion leads to only single VC is required for the HFBN(2, 1, 1). On there hand, off-chip HFBN are constructed with the 2DTorus network arrangement. Hence, it requires 2 VC for its deadlock-free. In summary. HFBN requires 2 VC for its deadlock-free routing. However, using the required number of VCs, we could like to consider a proof for the HFBN deadlock-freeness. To prove the deadlock-free of the proposed routing algorithm, the routing path are divided into multiple states.

- **State 1:** Transfer of packet from the source core to outlet core of the Intra-BM.
- **State 2:** Transfer of packet for higher level.

  State 2.i.1: Transfer of packet to the outlet core of Level (L - i) through the y-link for the Intra-BM.

  State 2.i.2: Transfer of packet of Level (L - i) through the y-link for the Inter-BM.

  State 2.i.3: Transfer of packet to the outlet core of Level (L - i) through the x-link for the Intra-BM.

  State 2.i.4: Transfer of packet of Level (L - i) through the x-link for the Inter-BM.
- **State 3:** Transfer of packet from the receiving core to the destination core of the Intra-BM.

Lemma 3.1: if a message is routed in the order $y \to x$ in a 2DMesh network, then the network is deadlock free with 1 VCs [38].

Lemma 3.2: if a message is routed in the order $y \to x$ in a 2DTorus network, then the network is deadlock free with 2 VCs [38].

Theorem 3.1: A HFBN is deadlock-free with 2 VCs.

Proof: The BM for HFBN(2, L, q) follows the flattened butterfly connection. Hence, at this network level requires only 1 VC, which is also proofed by the lemma 3.1. However, the higher level network is designed with toroidal connection. Hence, it requires 2 VCs for the upper level deadlock free and even if we considered the routing phase phase-1 and phase-3 requires 1 VCs for HFBN(2, L, q). However, phase-2 with the sub-phases considered the toroidal connectivity. Hence, the HFBN requires 2 VCs for this case. Finally, in summary, HFBN is deadlock-free with the two virtual channels.

3.3 Self Evaluation on HFBN

3.3.1 Optimal Configuration of HFBN for m, L and q

Optimal configuration of HFBN for choosing m and L can determined by the algorithm 3, where the value of the m and L can be easily obtained for the optimal configuration. However, the number of cores for HFBN is predefined, which is defined as $N = 2^{2mL}$.

Moreover, with the same core number HFBN has multiple possible configuration. For example- HFBN(2, 4, 1) has 65,536 cores and even HFBN(4, 2, 15) and HFBN(8, 1, 1) have also 65,536 cores. In such case, algorithm 3 can be used to choose the optimal configuration for HFBN with m and L. This algorithm confirms the possibility of the maximum network level through the minimum m, which makes sure the maximum upper level connectivity for the optimal outcome. In addition, we have also considered the traffic analysis to verify the optimal outcome (DCP graphs are considered with the data flits only as the accepted throughput). Table 3.7 and Table 3.8 (Each intra-chip is considered with 16 cores (L1 Links showed in Table 3.6), up to 256 cores for each inter-chip (excluding the intra-chip connectivity) (L2 Links showed in Table 3.6) and outer of inter-chip level connectivity, each link assumed as the optical link connectivity (However, HFBN(3,4,4) is considered with 16 cores as the intra-chip module and up to 64 cores as the each inter-chip module and outer links from the inter-chip module is treated as the optical links)) are used respectively for the various analysis. We have considered the static link power for this analysis along with the same inter-rack and intra-rack link power for reducing the complexity in power analysis. Figure 3.10 shows the NEU and Figure 3.11 shows the performance analysis of 16M cores (Table 3.9 shows the evidence results for this analysis). On the other hand, Table 3.10 shows the simulation condition for the optimal configuration of m, L, q with 1M cores (same link power usage (Table 3.8) is also been considered for the 1M core analysis). Figure 3.12 shows the network energy usage and finally, Figure 3.13 shows the traffic analysis for this case (Table 3.11 shows the evaluation results for 1M analysis). Moreover, we have also showed the optimal configuration analysis with 65K cores with same power parameters as Table 3.8 and considered traffic parameters is showed in Table 3.12. Figure 3.14 shows the NEU analysis with various configurations for 65K cores. Those graphs leads to three possible scenarios-

- **Same m, Different q**: This scenario can be describe by the Figure 3.15, where HFBN(4, 2, 15) and HFBN(4, 2, 30) is showed and the performance of HFBN(4, 2, 30) is much better than the HFBN(4, 2, 15). However, increasing the q, increases the power usage as well as the NEU (showed in Figure 3.14, where HFBN(4, 2, 15) shows much low NEU than the HFBN(4, 2, 30) at the zero load latency).

- **Same q, Different m**: This scenario can be describe by the both 1M and 65K analysis, where HFBN(10, 1, X) shows much worst result than the HFBN(2, 5, 1) in case of 1M analysis and HFBN(8, 1, X) shows also the worst zero load latency and the saturation rate over the HFBN(2, 4, 1) in case of 65K analysis.

- **Low m**: Figure 3.10 shows the evidence of 16M analysis through the superiority of HFBN(2, 6, 1) over the HFBN(3, 4, 4). In addition, Figure 3.15 shows the 65K cores traffic analysis, where for HFBN(2, 4, 1) with low m = 2, shows better zero load latency and the much better saturation rate with respect to large m (HFBN(4, 2, 15) or HFBN(4, 2, 30)).

Algorithm 3 Optimal Configuration of HFBN for m and L

```
$OptimalConfiguration\_Selection(NODEN)$; // NODEN is the total core number
        L = 0, m = 0;
        for i = 2; $2^i * 2^i * 2^{2i}$ < = NODEN; i++
         for j = 1; j < = 2 * ($2^i$ - 1) + 1 ; j++
           if (L < j and $2^{2ij}$ = NODEN), then L = j; m = i; endif;
          endfor;
        endfor;
        return (m, L);
end
```

Table 3.7: Simulation Environment for Traffic Analysis (16M cores)

Parameter	Value	Units
Cores	16,777,216	-
Flow Control	wormhole switching	-
Channel Buffer Size	4	-
Simulation Cycle	500	Clock cycles
Virtual Channels	8	-
Traffic Pattern	Uniform Traffic	-
Router Pipeline Cycle	1	clocks
link Latency	1	clocks
Packet size	12	flits

Table 3.8: Consideration for Power Usage

Power Evaluation	Value	Units
Electrical link		
Intra-chip Link Power [49]	0.0012	watt
Inter-chip Link Power [49]	0.15	watt
Optical link (inter-rack and intra-rack links)		
GBIC Module [FG-TRAN-SFP28-SR [45]]	1.2	watt
Intra-rack Link Power [49]	0.035	watt
Inter-rack Link Power [49]	0.035	watt

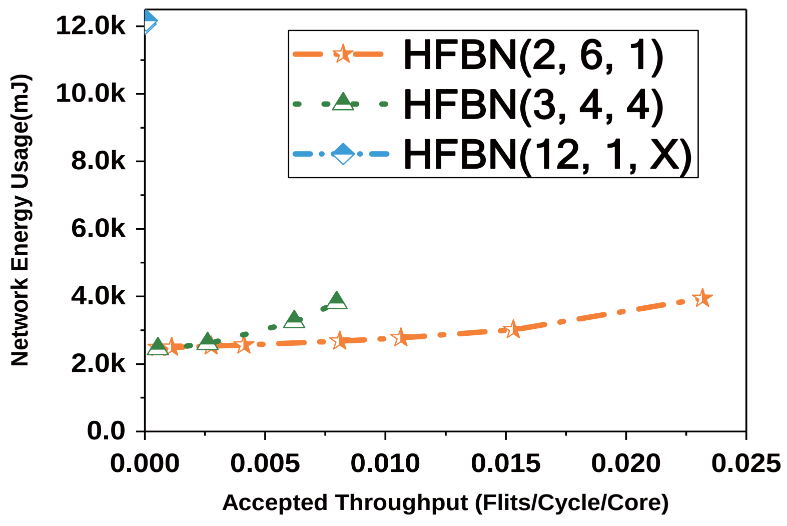


Figure 3.10: Analysis on NEU of HFBN for m, L, q(16M Cores)

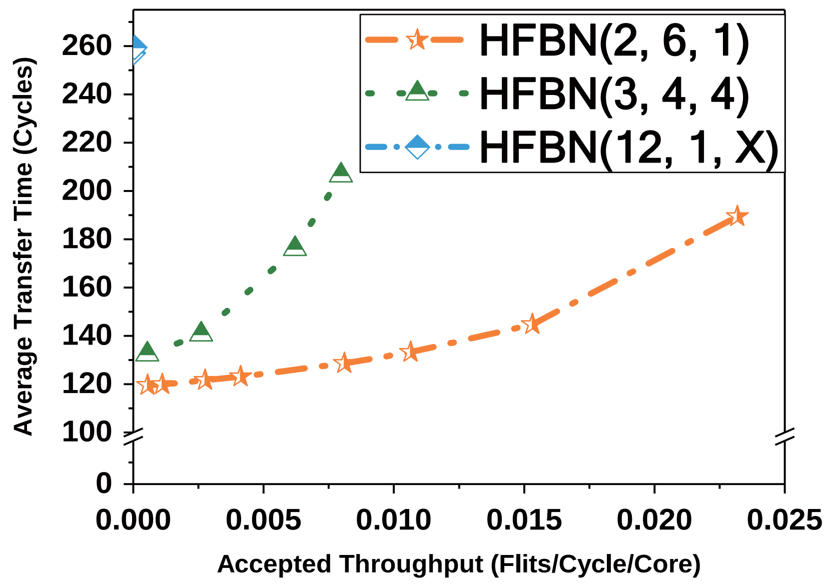


Figure 3.11: Analysis on Uniform Traffic of HFBN for m, L, q(16M Cores)

Table 3.9: Uniform Traffic Energy Analysis(16M cores)

Accepted Through-put(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Through-put(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,6,1)			HFBN(12,1,X)		
0.000055	119.52	20,801,231.10	0.000005	257.09	46,933,003.40
0.001110	119.94	20,801,231.10	0.000023	258.06	46,933,003.40
0.004120	123.14	20,801,231.10			
0.008100	128.64	20,801,231.10			
0.010640	133.22	20,801,231.10			
0.023180	189.37	20,801,231.10			
HFBN(3,4,4)					
0.000536	132.31	18,465,423.30			
0.002610	140.62	18,465,423.30			
0.006210	175.95	18,465,423.30			
0.007976	206.45	18,465,423.30			

Table 3.10: Simulation Environment for Traffic Analysis (1M cores)

Parameter	Value	Units
Cores	1,048,576	-
Flow Control	wormhole switching	-
Packet Size	6 (+ 6 Header flits)	flits
Channel Buffer Size	4	-
Simulation Cycle	5000	Clock cycles
Virtual Channels	6	-
Traffic Pattern	Uniform Traffic	-
link latency	1	clocks
Router Pipeline Cycle	1	clocks

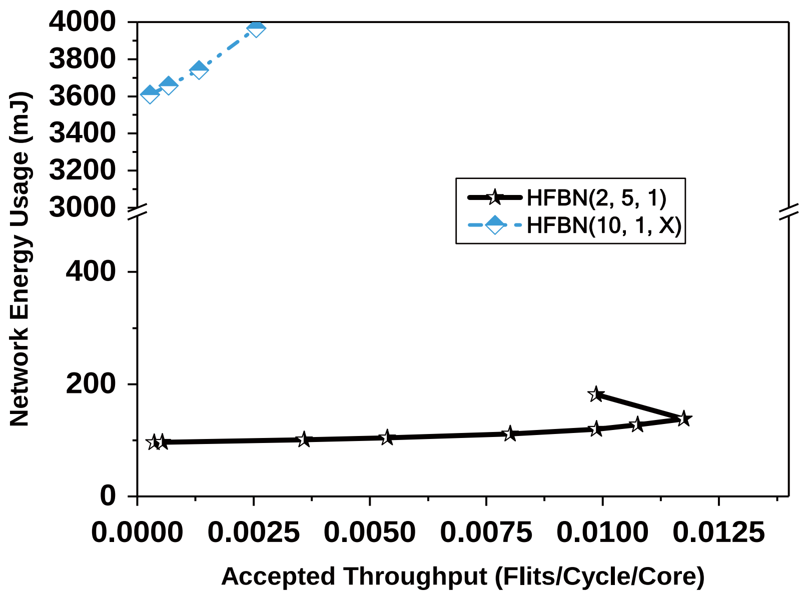


Figure 3.12: Analysis on NEU of HFBN for m, L, q(1M Cores)

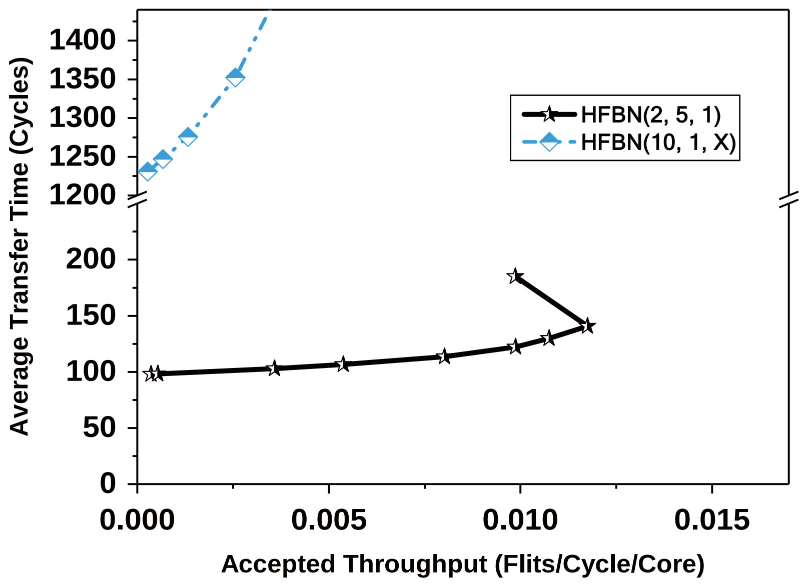


Figure 3.13: Analysis on Uniform Traffic of HFBN for m, L, q(1M Cores)

Table 3.11: Uniform Traffic Energy Analysis(1M cores)

Accepted Through-put(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Through-put(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,5,1)			HFBN(10,1,X)		
0.00036	98.04	980,916.63	0.00027	1230.76	2,933,312.717
0.00054	98.24	980,916.63	0.00067	1247.09	2,933,312.717
0.00359	102.93	980,916.63	0.00133	1275.54	2,933,312.717
0.00537	106.61	980,916.63	0.00256	1352.61	2,933,312.717
0.00801	113.65	980,916.63	0.00363	1456.16	2,933,312.717
0.00987	122.24	980,916.63			
0.01075	129.88	980,916.63			
0.01175	140.67	980,916.63			
0.00987	184.91	980,916.63			

Table 3.12: Simulation Environment for Traffic Analysis (65K cores)

Parameter	Value	Units
Cores	65,536	-
Packet Size	6 (+ 6 Header flits)	flits
Channel Buffer Size	2	-
flow Control	Wormhole	-
Simulation Cycle	5000	Clock cycles
Traffic Pattern	Uniform Traffic	-
Virtual Channels	4	-
link latency	1	clocks
Router Cycle	1	clocks

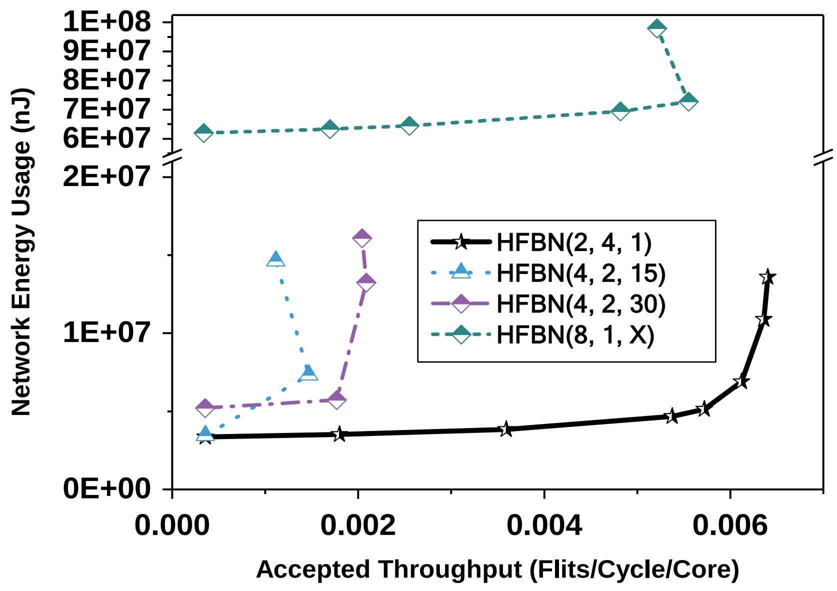


Figure 3.14: Analysis on NEU of HFBN for m, L, q(65K Cores)

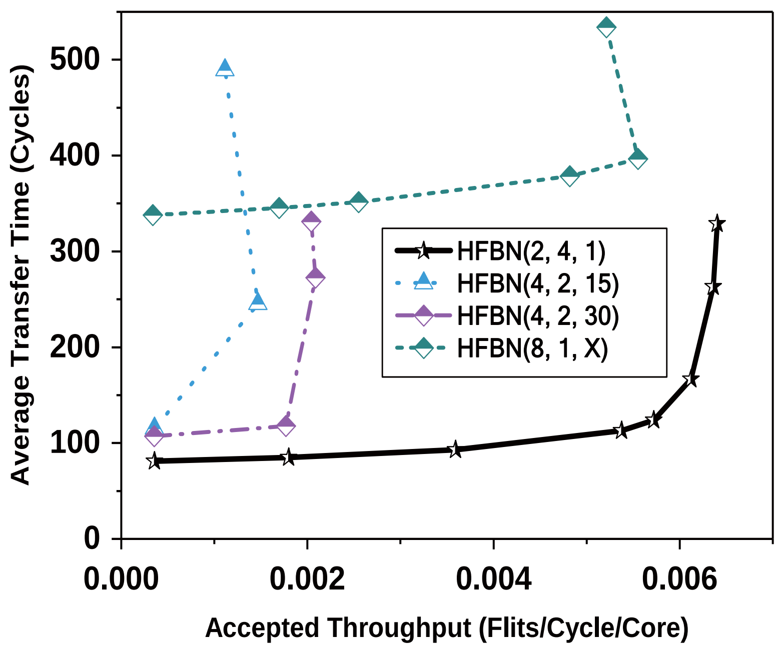


Figure 3.15: Analysis with Uniform traffic of HFBN for m, L, q(65K Cores)

Table 3.13: Uniform Traffic Energy Analysis(65K Cores)

Accepted Throughput(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Throughput(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,4,1)			HFBN(8,1,X)		
0.00035	81.19	41,359.76	0.00033	337.91	183,332.04
0.00180	84.91	41,359.76	0.00170	345.37	183,332.04
0.00359	92.89	41,359.76	0.00255	351.45	183,332.04
0.00538	113.01	41,359.76	0.00482	378.41	183,332.04
0.00572	124.02	41,359.76	0.00555	396.63	183,332.04
0.00612	166.76	41,359.76	0.00521	533.91	183,332.04
0.00636	263.35	41,359.76			
0.00640	328.78	41,359.76			
HFBN(4,2,15)			HFBN(4,2,30)		
0.00035	115.31	29,877.96	0.00035	107.24	48,578.76
0.00147	245.14	29,877.96	0.00177	117.74	48,578.76
0.00112	489.30	29,877.96	0.00209	272.68	48,578.76
			0.00204	331.27	48,578.76

Maximum effectiveness of paired connectivity for HFBN can be confirmed by Q_{max}. Q_{max} defines the maximum paired connectivity is possible for particular HFBN network level. For example, ($Q_{max} = 2(2^m - 1)$) if m = 2, $Q_{max} = 2(2^2 - 1) = 6$ leads to 6 possible paired connectivity, which also constitutes $L_{max} = 2$. In such case, the value of q is up to Q_{max}, which is an IDEAL_HFBN(2, 2, 6). However, if the q = 5 or q = 4, the configurations leads to the PARTIAL_HFBN(2, 2, 5) or PARTIAL_HFBN(2, 2, 4). However, for over than 1M cores q must be equal to 1 (Table 3.4 shows this configuration), if we consider m = 2. This consideration leads to choose the fixed configuration of HFBN(2, L, 1) for the entire thesis.

3.3.2 Evaluation on Deadlock-freeness

In this section, we also showed the deadlock-freeness of HFBN(2, 3, 1) through the real simulation. Table 3.14 shows the simulation conditions for obtaining the deadlock-free dynamic routing. Moreover, figure 3.16 shows the packet routing with respect to virtual channel 2, 4 and 6. Obviously, a large number of virtual channel helps to obtain better dynamic performance; however, it also increases the network power usages. Hence, the consideration for a network should maintain a low number of virtual channels for its deadlock-freeness. Figure 3.16 also ensures that HFBN(2, 3, 1) requires only 2 VC for becoming deadlock-free.

Table 3.14: Simulation enviornment for HFBN

Parameter	Value	Units
Cores	4,096	-
Message Size	16 (+2 Header flit)	flits
Flow Control	wormhole switching	-
Channel Buffer Size	2	-
Simulation Cycle	20,000	Clock cycles
Traffic Pattern	Uniform Traffic	-
Link Latency	1	clocks
Router pipeline cycle	1	clocks

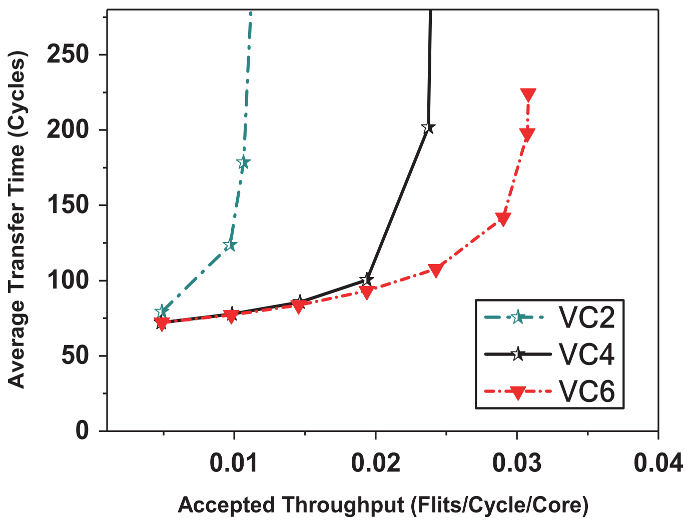


Figure 3.16: Uniform Traffic Analysis for Deadlock-freeness

3.4 Summary

In this chapter, we have mainly defined the network architecture for HFBN(m, L, q). This chapter also ensures that why and how we have considered the particular network architecture for the on-chip as well as the off-chip level. Then, we ensure the network definition of HFBN to reduce the confusion in designing this network along with the link connectivity at the on-chip and off-chip level. After that routing algorithm 2 shows the dimension-order routing for HFBN as well as the number of virtual channels are required for the deadlock-free routing for HFBN are mentioned in Section 3.2.6. We have also showed the algorithm to find the optimal m, L and q (algorithm 3).

Chapter 4

Efficient Energy Usage

4.1 Introduction

Modern supercomputers consider the performance constraints as the primary factor in selecting the suitable network architecture. To gain high network bandwidth, low latency and high network throughput, high degree networks are the preferable choice. However, performance improvement can be achieved through the increased number of CPU cores and low degree networks can provide better energy efficiency through the low power usage. For example- Tofu is a 6D mesh/torus network, used in K-computer [26] can provide more than 10 petaflops performance with requiring about 12.6MW of electrical power. On the other hand, IBM Sequoia achieved the Linkpack performance of 17.17 petaFlops, about 63% faster than the K-computer with 123% more cores and consumes about 7.9MW, 37% less than the K-computer, considering the 5DTorus network Sequoia ensures more energy efficiency over the K-computer's 6D network [61].

4.2 Evaluation on Efficient Energy Usage

Efficient energy usage is an important consideration for MPC systems. As the modern MPCs are highly affected by the power consumptions, efficient energy usage can able to trace the system performance with respect to power usage which will be a key feature in the field of interconnection networks. The choice of this parameter has been taken from the observation of scenario where network with little poor performance, but having a much better power efficiency, had always been rejected. Efficient energy usage is treated as the goal for reducing the amount of power usage which is required to maintain the suitable network performance. Regarding the definition of performance, it can be considered as the dynamic network performance of the corresponding network and the power usage leads to network power usage. In this section, all the DCP graphs are considered with the data flits only as the accepted throughput. Equation 4.1 shows the consideration to obtain the network energy usage. Here, we have considered the single clock cycles as 1ns (as the system clock is 1GHz). Hence, network energy usage leads to average flits transfer time and the total power usage for transmitting the flits. On the other hand, efficient energy usage is the reduction of obtained network energy usage in comparing between two networks with the relative request-probability (r). In DCP analysis, packets are transmitted by the request-probability (r) during the simulation clock cycles. We have considered a wormhole simulator for the DCP analysis, which is specially designed for the hierarchical networks [40]. On the other hand, electrical power is considered up to inter-chip level (256 cores) and we have used the Orion energy model for this analysis [33]. Electrical power analysis considered with the various traffic patterns (used the Garnet 1.0 simulator for traffic generation [34] (used the default table-based routing)). Now, this analysis gives required power usage for single electrical module. Hence, to obtain the total electrical power, we have multiplied the single electrical module power with total number of electrical modules. And then, to obtain the optical power, we have considered the fixed data driven power of intra-rack link (0.0101 watt) [49, 62] and inter-rack link (0.035 watt) [49] along with the per GBIC module power (FG-TRAN-SFP28-SR (1.2 watt) [45]) for the optical off-chip connectivity. Total optical power usage can be obtained from the multiplication of required number of optic links (intra-rack and the inter-racks) with its power usage and the required number of GBIC module with its power usage. Estimation of power usage has been showed later part of this chapter. In this section, we have considered with three traffic patterns analysis with various simulation conditions with respect to the number of computing cores.

$$\text{Network Energy Usage (NEU)} = \text{Average Transfer Time} \times \text{Network Total Power Usage} \tag{4.1}$$

As the high degree networks requires large number of off-chip interconnect, it's not suitable for large MPC systems mainly due to the required power usage. Large-scale analysis is considered with 2 possible case of 65K core analysis and the 1M core analysis. In both the analysis, we have considered various traffic patterns for the result evaluation. This section evaluates the energy analysis of Tofu network. However, the basic module (on-chip) for Tofu network considers 12 cores. And hence, we can consider 240 cores for the inter-chip level and the total number of cores for Tofu is considered with 4080 cores for the single rack level (for 65K the total number of core for Tofu network is 65,280 cores and 1M analysis is consist of 1,044,480 cores), whereas rest of the networks considers 256 cores for there inter-chip level and 4096 cores as the total number of simulating cores for each rack. In case of RTTM network, we have considered a = 4 and each upper level network is constructed with (4×4) 16 lower level sub-networks with twisted torus connectivity to have the same number of cores for the each possible case. In this section, we have also considered the static NEU analysis based on the average distance and the total link power usage (this power model is considered from Table 3.8). Static NEU is the multiplication of average distance and the total power usage. This analysis (Table 4.1) ensures that with the increase of network size the static NEU will have an huge impact over the network (specially, the network power usage will cause the large difference).

Table 4.1: Static NEU Evaluation

	Average Distance	Total Power Usage (W)	Static NEU
HFBN(2,5,1)	16.9195	980,916.63	16,596,618.90
HFBN(2,4,1)	13.0994	41,359.76	541,788.04
HFBN(2,3,1)	9.2815	1,338.26	12,421.11

4.3 Dynamic Communication Performance

Network performance heavily depends on the variable traffic patterns. And, even the running applications on a MPC system are heavily affected by the traffic patterns. Dynamic Communication Performance for networks considers various traffic patterns and is characterised by the latency and throughput. Latency refers to the time of a single packet to reach destination core from its source core. On the other hand, Network throughput is the rate at which packets can be delivered by the network. It refers to the maximum amount of information delivered per unit of time through the network. Latency can be defined by the below equation-

$$T = T_h + L/b, T_s = L/b \tag{4.2}$$

Here, T_h is the header latency, is the time for the header of the message to traverse the network. T_s is the serialisation latency, is the time for a packet of length L to cross a channel with bandwidth b.

4.3.1 Definition of Various Traffic Patterns

Network load has high effective influence on performance. Traffic patterns are responsible for the choice of particular source and destination core in any network. Traffic patterns can be random or non-random selection. Here, we have used the following non-uniform traffic patterns along with the uniform traffic patterns for dynamic communication performance (DCP) analysis.

Uniform- In the uniform traffic pattern, every core sends message to every other core with equal probability, i.e., source and destination are randomly selected for each generated message.

Perfect Shuffle- This pattern is defined as the fixed source-destination pair for every message. The core with binary value $a_{n-1}, a_{n-2}, ..., a_1, a_0$ communicate with $a_{n-2}, a_{n-3}, ...$ ...., a_0, a_{n-1} (rotate left 1 bit).

Bit-compliment- Fixed source-destination pair for every message. This case each core sends message to a such core with one's complement of its own address.

4.3.2 Evaluation on Dynamic Communication Performance

Dynamic communication performance (DCP) considers the latency and throughput of each network. Hence, dynamic communication performance shows the network zero load latency, saturation load and maximum amount of packet can be delivered per unit of time through the network. Accepted throughput(Flits/Cycle/Core) is the number of flits has been received at the destination cores with respect to total number of cores and the total simulation cycle. Here, DCP graphs are considered with the data flits only as the accepted throughput. On the other hand, average transfer time (measured in clock cycles) is the average delivery period for all the delivered packets within the given simulation time. To evaluate the dynamic communication performance HFBN, we considered a special designed simulator [40]. This simulator is specially designed for the hierarchical networks with the facility of explicitly changing the packet ID with the change of source router. Hence, the choice of particular virtual channel is possible for different links in making the network as deadlock-free. HFBN(2, L, q) requires only 1 VC for its deadlock-free routing. However, the off-chip level of HFBN network requires 2 VCs for its deadlock-free due to its torus connections. HFBN network considers the DOR routing for its dynamic performance. And even we also considered consider simple dimension-order routing rest of the networks with wormhole flow control. We have considered the intel compiler with mcmodel(= medium) for the 1M and 16M core analysis and for 65K cores analysis, DCP simulation results are obtained from Visual C++ 2017 compilation.

Flow control is responsible for the allocation of resources in packets. In DCP analysis, packet is the key component to ensure the network capability, who follows a certain route for reaching the destination core. The key resources in networks are the channels and buffers. Channels make sure the network connectivity and buffers are used to holding the packets temporarily at the cores. In the DCP analysis, we have considered the wormhole flow control. Wormhole flow control requires low buffering and most importantly, ensures the latency independence over the message distance. In wormhole routing each message is divided into packets, which are later divided into flits. Flits are consisted of two parts as header flits and data flits. Header flit holds the routing information and data flit follows the header flit through the network. On the other hand, DCP analysis considers only the deterministic routing for each network. Deterministic routing is also called as the oblivious routing. In deterministic routing, same routing path will always be considered between the same source and destination pair even though multiple routing path exists.

4.4 Estimation of Power Consumption

Power consumption is definitely the major concern for the exa-scale systems. Modern supercomputers are heavily affected by the on-chip as well as of off-chip power usages. One of the powerful supercomputer Tianhe-2 requires 24MW of electrical power in achieving 33.86 petaFlops performance with more than 3 million of core [44]. Hence, the required power at the exa-scale level will be similar to a single nuclear power plant, which is definitely unrealistic.

4.4.1 Assumption for the Power Model

The power consumption for the MPC system depends on various components such as network system, processor, memory module and for the cooling system. On the other hand, the network has a high impact on the total power usage. For example- 16-tile MIT RAW on-chip network consumes about 36% of the total chip power [46]. Hence, on-chip power is obviously the most important factor for estimating the total power usage. On the other hand, the required power at the rack level for each link is typically about 1W, bandwidth is over GB/s and cost is very high [47]. H. Wang and et. al. shows the power comparison for high speed electrical and optical interconnects for the interchip communication [48]. Hence, off-chip power estimation is also required for the analysis of total power usage. In this thesis, we have considered the fixed data driven power of intra-rack link (0.0101 watt) [49] and inter-rack link (0.035 watt) [49] along with the per GBIC module power (FG-TRAN-SFP28-SR (1.2 watt) [45]) for the optical off-chip connectivity. Hence, our power model is considered with the electric power at the inter-chip level and the optic power at the intra-rack or at the inter-rack level.

$$\begin{aligned} P_{total} &= P_{electrical} + P_{optical} \\ P_{electrical} &= P_{router} + P_{link} + P_{clock\ power} \\ P_{optical} &= P_{optical\ link} + P_{GBIC\ module\ power} \end{aligned} \tag{4.3}$$

4.4.2 Electrical Power Model

Our power model is based on the Orion energy model [33] using 65nm fabrication process. We have used GARNET 1.0 NoC simulator [34] for analysing dynamic power consumption. The power usage for the inter-chip network depends on the dynamic and leakage power usage. The router power is based on the router buffers, its local and global arbiter and the crossbar traverse. The dynamic energy model of router is being considered with Equation 4.4, where C is the capacitance, V is the supply voltage and finally α is the switching factor [33]. On the other hand, channels dynamic power are caused by the charging and discharging of capacitive loads, which is formulated as $P_{dy\_link} = \alpha C_1 V_{dd} f_{clk}$, where C_1 is the load capacitance, V_{dd} is the supply. We have considered 6 header flits along with the 6 data flits for the electrical power analysis. The header flits are merged with the data flits (8 bits for each data flits and 8 bits for each header flits). Hence, the total number of flits for the electrical power usage is considered as 6 flits for per packet. And, to obtain the total electrical power, we have multiplied the obtained simulated power usage with the total number of inter-chip module. The routing for this inter-chip analysis is considered with the default "Table-based" routing of the garnet 1.0 simulator.

$$E = 1/2\alpha C V^2 \quad [33] \tag{4.4}$$

4.5 Efficient Energy Usage Analysis (1M Cores)

1M cores are considered up-to the Level-5 network of HFBN(2,5,1). This case starting with the Table 4.2 parameters for the 1M traffic analysis. This case we evaluated the power simulation with the same traffic condition with respect to Table 4.3 and Table 4.4. Power analysis is obtained from parameters showed in Table 4.3 with the same accepted throughput (used as the injection rate in Garent 1.0 simulator [34]), which is considered as the parameter for dynamic communication performance. This section, we have also considered 256 cores for their inter-chip level and 1M cores as the total number of simulating cores.

Table 4.2: Simulation Environment for Traffic Analysis

Parameter	Value	Units
Cores	1,048,576	-
Flow Control	wormhole switching	-
Packet Size	6 (+ 6 Header flits)	flits
Channel Buffer Size	4	-
Simulation Cycle	5,000	Clock cycles
Virtual Channels	6	-
link latency	1	clocks
Router Pipeline Cycle	1	clocks

Table 4.3: Simulation Environment for Inter-chip Model (electrical interconnect)

Parameter	Value	Units
Fabrication process	65	nm
Average link length	25	[mm]
Operating frequency	1×10^9	Hz
Transistor type	HVT	-
Supply voltage	1.0	V
Packet Size	6	flits
Simulation cycle	5,000	-
Virtual Channel	6	-
VA Model	VC_select	-
VA Buffer Model	SRAM	-
Buffer Size	4	-
CLOCK PIPELINE STAGE	2	-

Table 4.4: Consideration for optical interconnect

Optical Power Evaluation	Value	Units
GBIC Module [FG-TRAN-SFP28-SR [45]]	1.2	watt
Intra-rack Link Power [49]	0.0101	watt
Inter-rack Link Power [49]	0.035	watt

4.5.1 Uniform Traffic

Now, energy usage evaluation for the 1M cores, Table 4.2 shows the traffic parameters for 1M cores evaluation and Table 4.3 shows the power parameter for electrical analysis (Table 4.4 is for the optical link power usage). And finally, Figure 4.1 shows the energy analysis for MMN, RTTM, 2DMesh, 2DTorus, HFBN, TTN and TESH network considering the the traffic analysis for 1M cores showed in Appendix A. This figure explains that HFBN can obtain about 23.49% efficiency over the TTN and comparing with the Tofu(40,32,68,3,2,2) network, HFBN can obtain about 87.26% at the zero load latency. This analysis also ensures that the zero load latency and the network saturation rate for HFBN is superior over any other networks. Table 4.5 shows the obtained latency and power usage for corresponding accepted throughput.

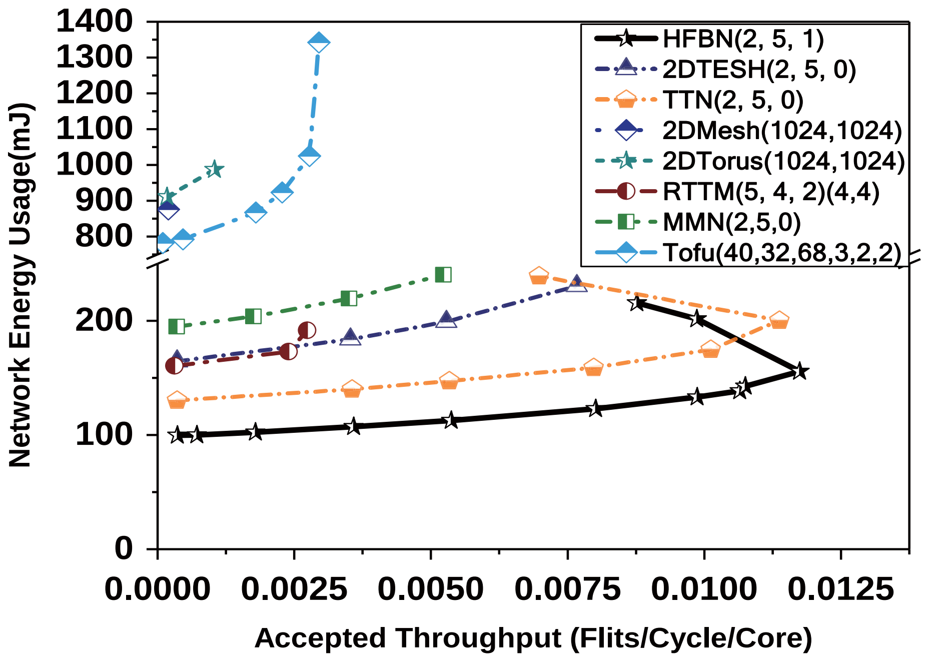


Figure 4.1: Energy Usage with uniform traffic (1M Cores)

Table 4.5: Uniform Traffic Energy Analysis(1M Cores)

Accepted Through-put(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Through-put(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,5,1)			TTN(2,5,0)		
0.00036	98.04	1,016,282.276	0.00035	129.52	1,005,457.695
0.00359	102.93	1,040,904.855	0.00356	135.70	1,031,891.050
0.00537	106.61	1,057,083.265	0.00534	140.31	1,049,271.349
0.00801	113.65	1,080,643.477	0.00798	148.94	1,066,662.592
0.00987	122.24	1,089,408.074	0.01012	159.89	1,093,031.072
0.01075	129.88	1,097,555.632	0.01138	181.72	1,102,476.112
0.01175	140.67	1,106,283.573			
2DTESH(2,5,0)			MMN(2,5,0)		
0.00035	164.20	1,002,150.429	0.00035	194.45	1,002,503.456
0.00353	177.38	1,036,368.351	0.00176	200.91	1,014,357.264
0.00528	188.04	1,058,399.641	0.00350	211.58	1,037,283.538
0.00767	213.28	1,080,497.549	0.00523	226.57	1,060,340.884
0.00840	267.55	1,090,882.183	0.00684	251.60	1,071,574.340
RTTM(5,4,2)(4,4)			2DTorus(1024,1024)		
0.00031	160.17	1,001,662.496	0.00017	2124.64	429,173.845
0.00240	169.09	1,022,779.390	0.00104	2225.95	443,548.466
0.00268	174.30	1,098,473.511			
Tofu(40,32,68,3,2,2)			2DMesh(1024,1024)		
0.00009	201.69	3,877,893.411	0.00019	2140.99	409,226.325
0.00180	223.20	3,885,843.705			
0.00228	237.20	3,893,086.482			
0.00278	263.33	3,893,086.482			

4.5.2 Perfect Shuffle Traffic

Figure 4.2 shows the energy usage for perfect shuffle traffic, which also ensures the superiority of HFBN over any other networks. At the zero load latency, 2DTorus network shows the worst energy usage among all the networks due to its high network latency (showed in Appendix A) and the high number of off-chip interconnect in compared to 2DMesh network. In this case, HFBN can sure about 47.07% better efficiency before the network saturation in comparing with the TTN. However, the Tofu(40,32,68,3,2,2) network shows the efficiency difference in about 86.32% in comparing with the HFBN at the zero load latency. Table 4.6 shows the obtained latency and power usage for corresponding accepted throughput for the perfect shuffle traffic.

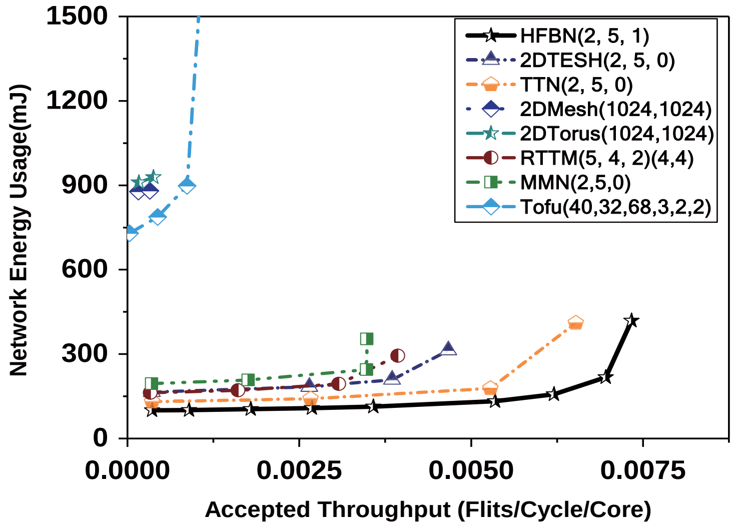


Figure 4.2: Energy Usage with Perfect Shuffle traffic (1M Cores)

Table 4.6: Perfect Shuffle Traffic Energy Analysis(1M Cores)

Accepted Through-put(F/C/C)	Average Transfer Time(C)	Total Power Usage(W)	Accepted Through-put(F/C/C)	Average Transfer Time(C)	Total Power Usage(W)
HFBN(2,5,1)			TTN(2,5,0)		
0.00036	98.24	1,016,344.396	0.00035	129.52	1,005,457.695
0.00268	104.49	1,033,150.924	0.00266	138.03	1,023,400.007
0.00535	125.53	1,057,554.473	0.00528	169.69	1,049,392.168
0.00620	147.13	1,065,559.210	0.00653	388.56	1,058,004.488
0.00696	204.21	1,065,559.218			
0.00734	389.91	1,073,596.948			
2DTESH(2,5,0)			MMN(2,5,0)		
0.00035	164.69	1,002,253.604	0.00035	194.27	1,002,652.195
0.00265	179.24	1,025,690.990	0.00176	204.70	1,015,397.640
0.00385	201.28	1,035,861.695	0.00348	235.84	1,037,633.993
0.00467	299.17	1,046,960.273			
RTTM(5,4,2)(4,4)			Tofu(40,32,68,3,2,2)		
0.00033	161.68	1,001,715.988	0.00003	188.22	3,877,428.878
0.00161	169.29	1,013,207.201	0.00044	203.05	3,877,428.878
0.00308	188.06	1,033,604.647	0.00086	231.59	3,877,428.878
0.00394	284.12	1,033,604.649	0.00105	397.42	3,884,556.609
2DTorus(1024,1024)			2DMesh(1024,1024)		
0.00016	2133.39	426,627.665	0.00016	2158.37	406,680.145
0.00037	2177.39	426,627.665	0.00032	2164.08	406,680.145

4.5.3 Bit-compliment Traffic

As the full system simulator (Gem5) has limited system scalability [51, 50], we have also considered the NAS parallel benchmarks [59] communication characteristics with the Message Passing Interface (MPI) implementation [58]. In static communication, compiled communication technique considers the compiler knowledge on application communication requirement and the provided network structure and allows to significantly optimize the performance of communications at the compile time [59]. The communication pattern of MPI programs can be sub-divided into three types: static communications, dynamic communications and dynamically analyzable communications. Static communications are those communications whose source and destination core are determined at the compile time. Dynamically analyzable selects source and destination core at runtime without incurring excessive overheads. Dynamic communications selects source and destination at only the runtime. However, majority of communications in scientific programs maintains the static communication. Hence, in this part of traffic analysis, we have considered the static communication pattern of MPI_Send, where selection of all source-destination pairs must be determined at compile time. We have considered the bit-compliment traffic pattern with fixed source and destination for this analysis. Table 4.2 shows the parameter consideration for the traffic analysis. And, Appendix A reviled the performance analysis with various networks along with the Figure 4.3 shows that HFBN can obtain about 30.76% better efficiency over the TTN in considering the bit-compliment traffic pattern. In case of Tofu, this difference leads to about 92.98% at the max saturation point of Tofu. Table 4.7 also reviles the obtained latency and power usage for corresponding accepted throughput for bit-compliment traffic.

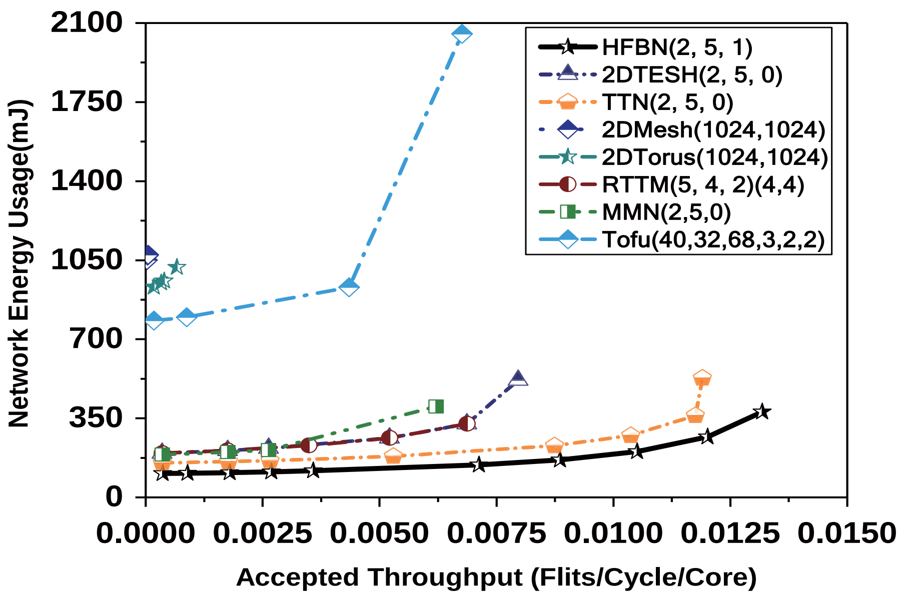


Figure 4.3: Energy Usage with Bit-compliment Traffic (1M Cores)

Table 4.7: Bit-compliment Traffic Energy Analysis(1M Cores)

Accepted Through-put(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Through-put(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,5,1)			TTN(2,5,0)		
0.00036	103.67	1,016,599.045	0.00035	151.39	1,005,353.181
0.00089	104.79	1,016,599.046	0.00178	155.75	1,014,532.475
0.00179	106.95	1,025,576.540	0.00266	159.12	1,022,844.130
0.00268	109.61	1,033,680.777	0.00530	173.99	1,048,342.117
0.00358	112.76	1,041,853.060	0.00874	213.06	1,073,310.903
0.00713	134.01	1,074,948.564	0.01038	251.67	1,090,219.065
0.00886	153.56	1,083,075.117	0.01176	329.88	1,098,682.066
0.01051	184.62	1,099,671.217	0.01191	481.18	1,098,682.067
0.01202	239.78	1,115,270.056			
0.01319	336.46	1,123,336.192			
2DTESH(2,5,0)			MMN(2,5,0)		
0.00035	194.96	1,002,485.325	0.00035	188.35	1,002,323.860
0.00176	204.47	1,014,682.012	0.00176	197.08	1,014,323.838
0.00263	212.48	1,025,684.290	0.00263	204.23	1,025,498.918
0.00522	248.48	1,059,694.237	0.00621	375.15	1,070,596.788
0.00687	304.23	1,071,195.862			
0.00796	479.40	1,081,820.547			
RTTM(5,4,2)(4,4)			2DTorus(1024,1024)		
0.00035	194.97	1,002,485.326	0.00017	2143.06	434,843.930
0.00176	204.47	1,014,680.604	0.00032	2183.46	434,843.930
0.00349	221.38	1,036,843.640	0.00067	2341.89	434,843.930
0.00522	248.56	1,059,693.225			
0.00688	304.20	1,071,195.174			
Tofu(40,32,68,3,2,2)			2DMesh(1024,1024)		
0.00017	201.87	3,878,017.259	0.00002	2530.79	414,896.410
0.00087	205.67	3,878,017.259	0.00003	2579.03	414,896.410
0.00435	238.24	3,906,982.387	0.00004	2588.11	414,896.410
0.00676	523.18	3,921,517.650	0.00004	2588.11	414,896.410

4.6 Efficient Energy Usage Analysis (65K Cores)

65K cores are considered up-to the Level-4 network of HFBN(2, 4, 1). This case starting with the Table 4.8 parameters for the 65K traffic analysis. This case we evaluated the power simulation with the same traffic condition with respect to Table 4.9 and Table 4.10. Power analysis is obtained from parameters showed in Table 4.9 with the same accepted throughput (used as the injection rate in Garent 1.0 simulator [34]), which is considered as the parameter for dynamic communication performance. This section, we have also considered 256 cores for there inter-chip level and 65,536 cores as the total number of simulating cores (other than Tofu network).

Table 4.8: Simulation Environment for Traffic Analysis

Parameter	Value	Units
Cores	65,536	-
Flow Control	wormhole switching	-
Packet Size	6 (+ 6 Header flits)	flits
Channel Buffer Size	2	-
Simulation Cycle	5000	Clock cycles
Virtual Channels	4	-
link latency	1	clocks
Router Pipeline Cycle	1	clocks

Table 4.9: Simulation Environment for Inter-chip Model (electrical interconnect)

Parameter	Value	Units
Fabrication process	65	nm
Average link length	25	[mm]
Operating frequency	1×10^9	Hz
Transistor type	HVT	-
Supply voltage	1.0	V
Packet Size	6	flits
Simulation cycle	5000	-
Virtual Channel	4	-
VA Model	VC_select	-
VA Buffer Model	SRAM	-
Buffer Size	2	-
CLOCK PIPELINE STAGE	2	-

Table 4.10: Consideration for optical interconnect

Optical Power Evaluation	Value	Units
GBIC Module [FG-TRAN-SFP28-SR [45]]	1.2	watt
Intra-rack Link Power [49]	0.0101	watt
Inter-rack Link Power [49]	0.035	watt

4.6.1 Uniform Traffic

In case of 65K analysis, we have considered electrical power analysis up-to inter-chip level and we considered the optical power for intra-rack level and inter-rack level. We have considered 6 header flits along with the 6 data flits and for the electrical power analysis. The header flits are merged the data flits (8 bits for each data flits and 8 bits for each header flits). Appendix A shows the traffic analysis for this case and using the Table 4.10 (optical connectivity) and Table 4.9 (electrical connectivity), we could obtain the power usage. Figure 4.4 shows the energy analysis for 65K uniform traffic, where HFBN could achieve about 22.24% better efficiency over the TTN. In comparing with the Tofu(20,16,17,3,2,2) network, even with the different network size of 65,280 cores, HFBN can also achieve much better efficiency (80.56%) with the low load latency. Table 4.11 shows the result evaluation of each throughput points showed in figure 4.4.

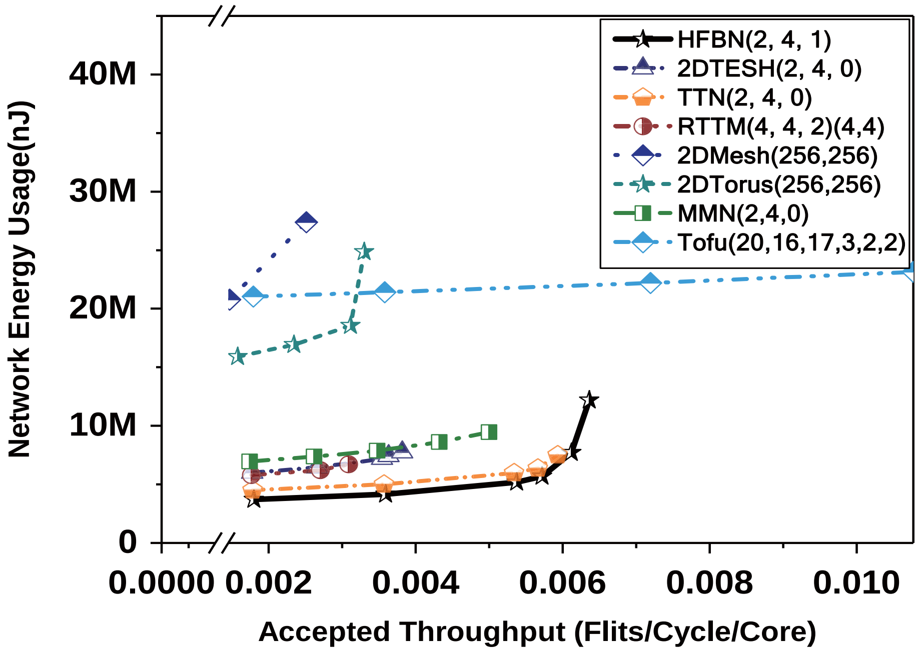


Figure 4.4: Energy Usage with uniform traffic (65K Cores)

Table 4.11: Uniform Traffic Energy Analysis(65K Cores)

Accepted Through-put(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Through-put(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,4,1)			TTN(2,4,0)		
0.00180	84.91	43,820.099	0.00178	104.29	43,253.880
0.00359	92.89	44,816.752	0.00357	112.68	44,324.910
0.00538	113.01	45,827.034	0.00534	131.51	45,408.380
0.00572	124.02	45,827.034	0.00567	140.04	45,408.381
0.00612	166.76	46,325.554	0.00593	165.40	45,408.381
0.00636	263.35	46,325.554			
2DTESH(2,4,0)			MMN(2,4,0)		
0.00178	138.93	43,241.635	0.00174	160.85	43,265.00
0.00266	146.97	43,934.866	0.00348	176.30	44,694.53
0.00355	160.92	44,637.352	0.00500	208.45	45,350.69
RTTM(4,4,2)(4,4)			2DTorus(256,256)		
0.0015	132.59	43,167.883	0.00158	668.73	23,778.818
0.0027	141.86	43,824.502	0.00312	731.06	25,408.434
0.0031	150.89	44,518.446	0.00330	978.12	25,408.434
Tofu(20,16,17,3,2,2)			2DMesh(256,256)		
0.00179	90.23	233,198.691	0.00147	922.50	22,532.102
0.00358	91.45	234,110.836	0.00251	1173.47	23,341.387
0.00720	94.09	235,890.004			
0.01077	97.5	237,280.525			

4.6.2 Perfect Shuffle Traffic

The zero load latency for HINs always ensures the lowest latency over the conventional networks. Perfect shuffle traffic analysis is also considered with the 65K cores and the same traffic parameters as the Table 4.8 for 65K analysis. In addition to traffic analysis parameters, the parameter used (Table 4.9 and 4.10) for the power analysis is also same as to analyze the network energy usage. Figure 4.5 shows the energy usage for Perfect Shuffle traffic, which also ensures the superiority of HFBN over any other networks. 2DMesh and 2DTorus networks show the worst efficiency among all the 2D networks due to its high network latency showed in Appendix A. This analysis ensures that HFBN has the superiority over TTN up-to 18.39% with the efficiency (Figure 4.5). Now, comparing with the Tofu(20,16,17,3,2,2) network, even with the different network size (65,280 cores), HFBN can achieve about 78.13% better efficiency with the low load latency. Table 4.12 also ensures the same comparison with various networks.

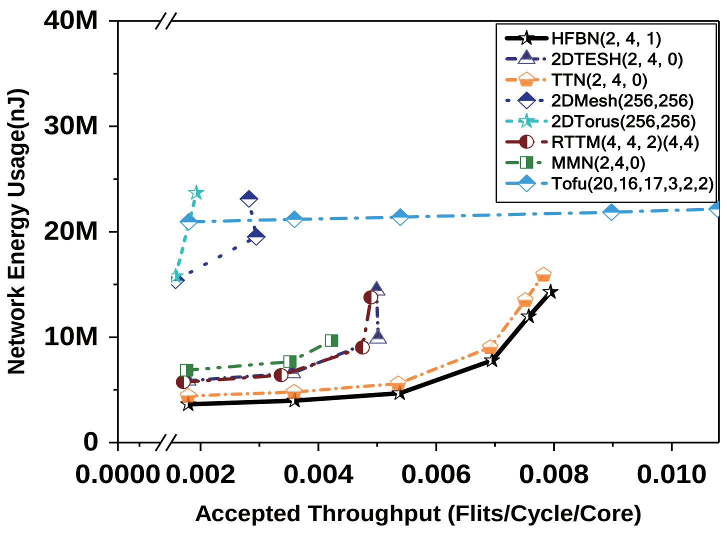


Figure 4.5: Energy Usage with Perfect Shuffle Traffic (65K Cores)

Table 4.12: Perfect Shuffle Traffic Energy Analysis(65K Cores)

Accepted Through-put(F/C/C)	Average Transfer Time(C)	Total Power Usage(W)	Accepted Through-put(F/C/C)	Average Transfer Time(C)	Total Power Usage(W)
HFBN(2,4,1)			TTN(2,4,0)		
0.00179	83.28	43,439.89	0.00179	102.47	43,262.49
0.00360	88.53	44,817.46	0.00359	108.55	44,305.26
0.00538	102.04	45,858.30	0.00536	122.99	45,416.70
0.00695	168.12	46,358.37	0.00692	196.56	45,954.92
0.00794	304.75	46,857.63	0.00782	342.49	46,470.30
2DTESH(2,4,0)			MMN(2,4,0)		
0.00178	135.51	43,269.02	0.00177	158.53	43,330.157
0.00356	147.64	44,606.47	0.00352	171.94	44,694.533
0.00502	213.78	46,057.97	0.00422	212.59	45,434.135
RTTM(4,4,2)(4,4)			Tofu(20,16,17,3,2,2)		
0.00171	132.68	43,193.14	0.00180	89.86	233,166.281
0.00337	143.91	44,465.82	0.00359	90.56	233,942.075
0.00475	199.70	45,133.54	0.00539	91.15	234,780.557
0.00489	304.75	45,133.54	0.01078	93.63	236,779.315
2DTorus(256,256)			2DMesh(256,256)		
0.00158	673.53	23,435.55	0.00157	694.16	22,188.834
0.00193	1011.06	23,435.55	0.00294	855.23	22,812.574

4.6.3 Bit-compliment Traffic

In case of zero load latency for HINs as well as the HFBN always ensures the lowest latency over the conventional networks. Bit-compliment traffic analysis is also considered with the 65K cores and the same performance analysis parameter as the Table 4.8 for 65K analysis. In addition to traffic analysis parameters, the parameter used (Table 4.9 and 4.10) for the power analysis is also same as to analyze the network energy usage. Figure 4.6 shows the energy usage for Bit-compliment traffic, which also ensures the superiority of HFBN over any other networks. 2DMesh and 2DTorus networks show the worst efficiency among all the 2D networks due to its high network latency showed in Appendix A. This analysis ensures that HFBN has the superiority over TTN up-to 22.36% with the energy efficiency (Figure 4.6). Now, comparing with the Tofu network, even with the different network size, HFBN can achieve much better energy efficiency (72.09%). Table 4.13 also ensures the same comparison with various networks.

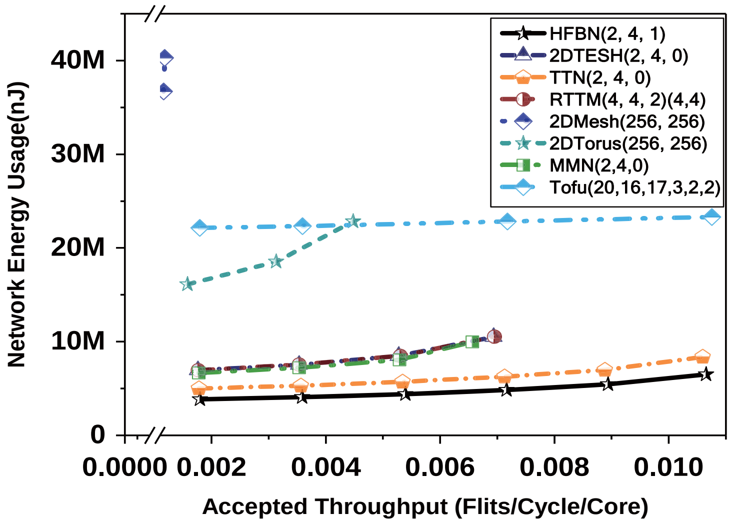


Figure 4.6: Energy Usage with Bit-compliment Traffic (65K Cores)

Table 4.13: Energy Analysis of Bit-compliment Traffic(65K Cores)

Accepted Through-put(F/C/C)	**Average Transfer Time(C)**	**Total Power Usage(W)**	**Accepted Through-put(F/C/C)**	**Average Transfer Time(C)**	**Total Power Usage(W)**
HFBN(2,4,1)			TTN(2,4,0)		
0.00180	87.38	43,863.22	0.00178	115.4	43,242.08
0.00359	90.81	44,879.06	0.00356	119.68	44,278.12
0.00539	95.85	45,926.69	0.00535	125.73	45,315.76
0.00716	103.26	46,942.44	0.00713	134.83	46,389.55
0.00894	114.86	47,446.79	0.00888	148.90	46,902.91
0.01066	134.19	48,480.92	0.01060	174.73	47,958.50
2DTESH(2,4,0)			MMN(2,4,0)		
0.00176	160.52	43,285.55	0.00178	153.84	43,263.26
0.00353	168.77	44,667.31	0.00353	161.52	44,630.33
0.00527	183.69	46,092.55	0.00529	175.20	46,052.64
0.00694	224.39	46,810.26	0.00656	213.23	46,759.02
RTTM(4,4,2)(4,4)			Tofu(20,16,17,3,2,2)		
0.00177	160.61	43,285.51	0.00179	94.94	233,190.43
0.00353	168.91	44,667.29	0.00359	95.38	234,073.92
0.00530	184.02	46,092.04	0.00718	96.76	235,820.15
0.00695	225.07	46,810.23	0.01076	98.29	237,161.66
2DTorus(256,256)			2DMesh(256,256)		
0.00158	658.43	24,467.09	0.00116	1580.6	23,220.37
0.00313	692.90	26,771.97	0.00118	1734.9	23,220.37
0.00448	820.16	27,834.23			

4.7 Summary

This chapter ensures the energy superiority of HFBN than any other networks. 65K core analysis showed that HFBN can achieved about 22.24% with uniform traffic and about 18.39% (with perfect shuffle traffic) better energy usage over the TTN. And finally, the case for 1M core ensures that HFBN shows the superiority at the zero load and also for the high saturation. This case HFBN can achieve about 23.49% better efficiency with uniform traffic, about 47.07% with perfect shuffle traffic and about 30.76% with bit-compliment traffic over the TTN.

Chapter 5

Conclusion

5.1 Conclusion

In this research plan, our main objective was to find an efficient interconnection network with requiring low power usage and can achieve high network performance for many-core processors. High degree networks are useful for performance considerations, but are not useful for low power usage. Hence, the target network focus on the efficient energy usage over achieving the suitable network performance.

Our new interconnection network is based on the 2D NoCs, renamed as the Hierarchical Flattened Butterfly Network (HFBN). Hence, this network has the high resiliency than the 3D networks (3D networks require higher switch surface area (0.1385) than the 2D networks (0.0924) [52]). On the other hand, HFBN with variable network size will require a fixed number of router radix (9); costing only about \$2,261.3 for each router [36]. In the static performance analysis, HFBN can show much better performance than conventional networks. For example- in comparing with 5DTorus network (used in Blue Gene/Q supercomputer), this network can also achieve about 32.5% better diameter performance (over 1M cores) and about 66.67% with over 200M cores, almost similar average distance performance and even shows much better cost analysis at 1M cores. On the other hand, in considering with the most recent high-performed hierarchical networks such as TTN, MMN, TESH, RTTM; this network confirms itself as an obvious choice. In case of diameter, HFBN can achieve about 34.15% better performance than TTN (with 1M cores); where in case of average distance, HFBN can achieve about 26.14% better performance.

Our energy evaluation considers various configurations starting from 65K cores to 1M cores. Most of the networks like- TTN, TESH, Mesh, Torus gets congested very quick. Hence, we couldn't evaluate the energy usage over 16M cores and compare them. However, starting with the 65K cores up to 1M cores HFBN shows its uttermost superiority over any other networks. 65K cores analysis showed that HFBN can achieved about 22.24% with uniform traffic and about 18.39% (with perfect shuffle traffic) better energy usage over the TTN. And finally, the case for 1M cores ensures that HFBN shows the superiority for zero load latency and also for the high saturation rate. This case HFBN can achieve about 23.49% better efficiency with uniform traffic, about 47% with perfect shuffle traffic and about 30.76% with bit-compliment traffic over the TTN. Comparing with the high degree network like- Tofu; HFBN also confirms its superiority. In case of 1M cores analysis of Tofu(40,32,68,3,2,2) network, HFBN can obtain about 87.26% better energy efficiency at the zero load latency with uniform traffic, about 86.32% with perfect shuffle traffic and about 92.98% with the bit-compliment traffic. On the other hand, with the 65K core analysis of Tofu(20,16,17,3,2,2) network, HFBN can obtain about 80.56% better efficiency with uniform traffic, 78.13% better efficiency with the perfect shuffle traffic and about 72.09% better efficiency with the bit-compliment traffic.

5.2 Future Research

This research considers about the energy efficient supercomputing for the next generation supercomputers. However, there is still lots of area where this research can be exploited. Some of the areas are discussed below-

1. Research on interconnection networks mainly conducted through network design, implementation, simulation analysis and experimental performance evaluation with collective communication patterns on various networks. In addition, in this paper, we have considered only the one-to-one communication. However, MPI (message passing interface) program requires to handle the multicast or broadcast message passing. This could be a very useful feature for future research on HFBN.

2. Exa-scale system requires large scalability. On the other hand, the failure rate trends to be increase vastly with the increase of system size. Thus, fault-tolerant capabilities could be very much beneficial. In the future research on HFBN, we would like to consider a new fault-tolerant routing algorithm such that it could handle the traffic in case of faulty nodes or any faulty links.

3. The prototyping approach will definitely reduce the required full scale implementation efforts. In the prototyping approach implemented through the Field Programmable Gate Array (FPGA), can also reduce the system cost and the risk of hardware implementation. Hence, our target for the next phase of this research also considers the implementation of HFBN on FPGA.

4. In this research, we also didn't consider the adaptive routing for the HFBN, where adaptive routing ensures better communication performance over the simple deterministic routing [40]. HFBN allows to travel the packet through numerous number of paths between any source and destination. On the other hand, deterministic routing follows a single path between any source to destination. If the selected path from source to destination is congested, the network traffic will be delayed. Thus, such fixed routing limits the network performance.

Bibliography

[1] Ajima, Y.; Inoue, T.; Hiramoto, S.; et. al. *The Tofu Interconnect*, Micro, IEEE, DOI. 10.1109/HOTI.2011.21, Vol. 32, pp. 21-31, JAN-FEB. 2012.

[2] Daniel Sanchez, George Michelogiannakis, et al. *An analysis of on-chip interconnection networks for large-scale chip multiprocessors*, Vol. 7, Issue 1, Article 4, April 2010.

[3] Cyriel Minkenberg,*http://www.systems.ethz.ch/*, IBM Research

[4] H. Fu and et. al., *The Sunway TaihuLight Supercomputer: System and Applications*, Science China Information Science, vol. 59 ed., July 2016.

[5] V. F. Pavlidis and E. G. Friedman, *3-D Topologies for Networks-on-Chip*, IEEE Trans. on VLSI syst., Vol. 15, Oct. 2007.

[6] The Green 500, Energy-efficient computers, http://www.green500.org/

[7] Swiss National Supercomputing Centre, *https://en.wikipedia.org/wiki/*

[8] C. Leiserson, Z. Abuhamdeh, D. Douglas, C. Feynman, M. Ganmukhi, J. Hill, W. Hillis, B. Kuszmaul, M. St. Pierre, D. Wells, M. Wong-Chan, Y. Saw-Wen, and R. Zak, *The network architecture of the Connection Machine CM-5*, Journal of Parallel and Distributed Computing, vol. 33, no. 2, pp. 145-158, 1996.

[9] R. Arlanskas, *iPSC/2 system: a second generation hypercube*, Proceedings of 3rd ACM Conference on Hypercube Concurrent Computers and Applications, pp. 38-42, 1988.

[10] S.F. Nugent, *The iPSC/2 direct-connect communication technology*, Proceedings Conference Hypercube Concurrent Computers and Applications, pp. 51-60, 1988.

[11] R.E. Kessler, J.L Swarszmeier, *Cray T3D: a new dimension for Cray research*, Proceedings CompCon, pp. 176-182, 1993.

[12] Cray Research Inc., The Cray T3E scalable parallel processing system, *http://www.cray.com/PUBLIC/product-info/T3E/CRAY_T3E.html.*

[13] M. Noakes, D.A. Wallach, and W.J. Dally, *The J-machine multicomputer: an architectural evaluation*, Proceedings of the 20th International Symposium Computer Architecture, pp. 224-235, 1993.

[14] M. Noakes and W.J. Dally, *System design of the J-machine*, Proceedings of Advanced Research in VLSI, pp. 179-192, 1990.

[15] Intel Corp., Paragon XP/S product overview, Supercomputer Systems Division, Beaverton, Oregon, 1991.

[16] D. Liu, Ch. Svensson, *Power Consumption Estimation in CMOS VLSI Chips*, IEEE Journal of SSC, June 1994.

[17] IBM Blue Gene/Q Architecture, *https://www.alcf.anl.gov/*

[18] Multistage interconnection networks,
https://en.wikipedia.org/wiki/Multistage_interconnection_networks

[19] Akash Punhani, Nitin and Pardeep Kumar, *A Modified Diagonal Mesh Interconnection Network*, INDICON, pp. 1-6, Dec. 2014.

[20] S. Konstantinidou and L. Snyder, *The Chaos Router, IEEE Transactions on Computers*, vol. 43, no. 12, pp. 1386-1397, 1994.

[21] V.K. Jain, T.Ghirmai, and S.Horiguchi, *TESH:A new hierarichical interconnection network for massively parallel computing*, IEICE Trans. on Inf. & Syst., vol. E80-D, pp. 837-846, 1997.

[22] Y. Liu, C. Li, and J. Han, *RTTM: a new hierarchical interconnection network for massively parallel computing. High performance computing and applications*, High Performance Computing and Applications, vol. 5938 of Lecture Notes in Computer Science, pp. 264-271, Springer, Berlin, Germany, 2010.

[23] M.M. Hafizur Rahman, Yukinori Sato and Yasushi Inoguchi, *High and stable performance under adverse traffic patterns of tori-connected Torus network*, Computers & Electrical Engineering, vol. 39, pp. 973-983, 2013.

[24] M.M. Hafizur Rahman, Susumu Horiguchi, *HTN: a new hierarchical interconnection network for massively parallel computers*, IEICE Trans. on Information and Systems, Vol. E86-D, pp. 1479-1486, 2003.

[25] Md. Rabiul Awal, M.A.H. Akhand, at.el., *A New Hierarchical Interconnection Network for Future Generation Parallel Computer* 16th International Conference Computer and Information Technology, Khulna, Bangladesh, March, 2014.

[26] K computer, Tofu interconnect, http://en.wikipedia.org/wiki/K_computer

[27] J. Hsu., *Power problems threaten to strangle exascale computing*, in IEEE Spectrum, http://spectrum.ieee.org/computing/hardware/power-problems-threaten-to-strangle-exascale-computing. (accessed on 2017-03-24), 2015.

[28] S. M. Hang-Sheng Wang, Li. S Shiuan Peh, *A power model for routers: modeling alpha 21364 and infiniband routers*, Proceedings of 10th Symposium on High Performance Interconnects, 2002.

[29] Blue Gene Supercomputer, *http://www.cs.cmu.edu/*

[30] I. Fujiwara, M. Koiuchi, and H. Casanova, *Cabinet layout optimization of supercomputer topologies for shorter cable length*, 13th International Conference on Parallel and Distributed Computing, pp. 227-232, 2008.

[31] M. J. Koop, W. Huang, K. G. krishnan, and D. Panda., *Performance analysis and evaluation of pcie 2.0 and quad-data rate infiniband*, 16th IEEE Symp. on High Performance Interconnects. pp. 85-92, 2008.

[32] I. M. Technologies, *Switched fabric, quad data rate*, https://en.wikipedia.org/wiki/InfiniBand, accessed 2016.04.21.

[33] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, *Orion 2.0: A power-area simulator for interconnection networks*, IEEE Transactions on very large scale integration (VLSI) systems, vol. 20, Issue. 1, pp. 191-196, 2011.

[34] N. Agarwal, T. K., L.-S. Peh, and N. Jha., *Garnet: A detailed on-chip network model inside a full-system simulator*, IEEE Int. Sym. on Perf. Analysis of Systems and Software, pp. 33-42, 2009.

[35] J. Kim, W. J. Dally, and James Balfour, *Flattened Butterfly Topology for On-Chip Networks*, 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), 1-5 Dec. 2007.

[36] Maciej Besta, Torsten Hoefler, *Slim Fly: A Cost Effective Low-Diameter Network Topology* SC14, November 16-21, 2014, New Orleans, Louisiana, USA

[37] J. Kim, W. J. Dally, and D. Abts, *Flattened butterfly: A cost-efficient topology for high-radix networks*, IEEE J. Solid-State Circuits, Vol. 43, No. 1. pp. 29-41, 2007.

[38] Topics in parallel computing, Virtual channels and deadlocks, *http://pages.cs.wisc.edu/ tvrdik/8/html/Section8.html*

[39] Al-Fares, Loukissas, Vahdat, *A Scalable, Commodity Data Center Network Architecture*, proceedings of SIGCOMM, 2008.

[40] Yasuyuki Miura, K. Shimozono, and S. Watanabe, *An Adaptive Routing of the 2-D Torus Network Based on Turn Model* Int. Symp. on Comp. and Net., pp. 587-591, Dec. 2013.

[41] E. John,Juan Rubio, Unique Chips and Systems, CRC Press, 2007

[42] S.R.Vangal, et. al., *An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS*, IEEE Journal of Solid-State Circuits, Volume 43, Issue: 1, Jan. 2008.

[43] 32 nanometer Wikipedia, https://en.wikipedia.org/wiki/32_nanometer/

[44] Tianhe-2 Supercomputer, https://en.wikipedia.org/wiki/Tianhe-2

[45] Fortinet Transceivers, GBic Module, FG-TRAN-SFP28-SR, https://www.fortinet.com/

[46] H. Wang, L. Peh, *Power-driven design of router microarchitectures in on-chip networks*, International Symposium on Microarchitecture (MICRO-36), Dec. 2003.

[47] I. M. Technologies, Switched fabric, quad data rate, in https://en.wikipedia.org/wiki/InfiniBand.

[48] H. Cho, P. Kapur, K. C. Saraswat, "Power Comparison Between High-Speed Electrical and Optical Interconnects for Interchip Communication", *Journal of Lightwave Technology*, DOI. 10.1109/IITC.2004.1345710, VOL. 22, NO. 9, September 2004.

[49] Keren Bergman, Silicon Photonics for Extreme Scale Computing, https://press3.mcs.anl.gov/

[50] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver., *Full-system analysis and characterization of interactive smartphone applications*, proceeding of the IEEE International Symposium on Workload Characterization (IISWC), pages 81-90, Austin, TX, November 2011.

[51] The gem5 Simulator, *http://www.gem5.org/Main_Page*.

[52] Brett Stanly Feero, Partha Pratim Pande, *Networks-on-chip in a three-dimensional environment: A performance evaluation*, IEEE Transactions on computers, Vol. 58, No. 1., 2009.

[53] William J. Dally, B. P. Towles, *Principles and Practices of Interconnection Networks*, Elsevier, Morgan Kaufmann Publications.

[54] Mostafa Abd-El-Barr, Turki F. Al-Somani, *Topological Properties of Hierarchical Interconnection Networks: A Review and Comparison*, Journal of Electrical and Computer Engineering Article ID 189434, vol. 2011, February 2011.

[55] S.P. Mohanty, B. N. B. Ray, S. N. Patro, A.R. Tripathy, *Topological Properties of a New Fault Tolerant Interconnection Network for Parallel Computer*, International Conference on Information Technology, pp. 36-40, Dec. 2008.

[56] Nibedita Adhikari, C. R. Tripathy, *The Folded Crossed Cube: A New Interconnection Network for Parallel Systems*, International Journal of Computer Applications, vol. 4, no. 3, July 2010.

[57] Dilip Sarkar, *Cost and Time-Cost Effectiveness of Multiprocessing*, IEEE Transactions on parallel and distributed systems, Vol. 4, No. 6, June 1993.

[58] The MPI Forum, The MPI-2: Extensions to the Message Passing Interface. *http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html, July, 1997.*

[59] A. Faraj, X. Yuan, *Communication Characteristics in the NAS Parallel Benchmarks*, https://pdfs.semanticscholar.org/.

[60] M. M. Hafizur Rahman, Y. Sato, et. al., *TTN: A High Performance Hierarchical Interconnection Network for Massively Parallel Computers*, IEICE TRANS. INF. & SYST., VOL.E92-D, NO.5, MAY 2009.

[61] IBM Sequoia, Petascale Blue Gene/Q Supercomputer, *https://en.wikipedia.org/wiki/IBM_Sequoia*

[62] Erik Agrell, Magnus Karlsson, A R Chraplyvy, David J Richardson, Peter M Krummrich, Peter Winzer, Kim Roberts, Johannes Karl Fischer, Seb J Savory, Benjamin J Eggleton, *Roadmap of optical communications*, Journal of Optics, Volume 18, Number 6, 3 May 2016.

[63] Yuuichirou Ajima, Tomohiro Inoue, Shinya Hiramoto, Toshiyuki Shimizu, *Tofu: Interconnect for the K computer*, FUJITSU Sci. Tech. Journal, Vol. 48, No. 3, pp. 280 - 285 (July 2012).

[64] Sun, Chen, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic, *DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling*, 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip (n.d.).

Publications

1. **Books and Published Articles in Scholarly Journals, etc.:**

   - M.M. Hafizur Rahman, Faiz Al Faisal, Nor R.M., Sembok T.M.T., Behera D.K., Inoguchi Y., "Cost Effectiveness Analysis of a Vertical Midimew-Connected Mesh Network (VMMN)", In: Behera H., Mohapatra D. (eds) Computational Intelligence in Data Mining. Advances in Intelligent Systems and Computing, vol. 556, pp. 43-53, Springer, Singapore, 2017.

2. **Expositions and Reviews in Scholarly Journals:**

   - Faiz Al Faisal, M.M. Hafizur Rahman, Yasushi Inoguchi, A new power efficient high performance interconnection network for many-core processors, Journal of Parallel and Distributed Computing, Volume 101, Pages- 92-102, March 2017.

3. **Presentations at International Academic Conferences (Reviewed):**

   - Faiz Al Faisal, M.M. Hafizur Rahman, Yasushi Inoguchi, "Power Analysis with Variable Traffic Loads for Next Generation Interconnection Networks", 18th IEEE Int. Conf. on High Perf. Computing and Communications (HPCC), Sydney, Australia, Dec. 2016.
   - Faiz Al Faisal, M.M. Hafizur Rahman, Yasushi Inoguchi, Dynamic Communication Performance of TTN with Uniform and Non-uniform Traffic Patterns using Virtual Cut-Through Flow Control, *International Conference on Advanced Computer Science Applications and Technologies (ACSAT2014)*, Pages-133-138, 29-30 Dec. 2014.
   - Faiz Al Faisal, M.M. Hafizur Rahman, Yasushi Inoguchi, Dynamic Communication Performance of STTN under various Traffic Patterns using Virtual Cut-Through Flow Control, IEEE 17th International Conference on Computational Science and Engineering (CSE), Chengdu, China, Pages-1804-1809, 19-21 Dec. 2014.
   - M.M. Hafizur Rahman, Faiz Al Faisal, Rizal Mohd. Nor, Tengku Sembok, M.A.H. Akhand, Packing Density and Message Traffic Density of a Midimew Connected Mesh Network, International Conference on Computer and Communication Engineering, Kuala Lampur, Malaysia, 2016.

4. **Presentations at Domestic Academic Conferences(Non-Reviewed):**

   - Faiz Al Faisal, M.M. Hafizur Rahman, Yasushi Inoguchi, Topological Analysis of Low-Powered 3D-TESH Network, The Institute of Electronics, Information and Communication Engineers, CPSY2015-126, Pp. 143-148, Tokyo, Japan, Jan. 2016 (Non-reviewed).

Appendix A

Large-scale Dynamic Communication Performance

The target for this large-scale analysis is to ensure that HFBN is capable of performing as the exa-scale supercomputer network. From this analysis, it is quite evident that HFBN is capable for the large-scale systems with its large network scalability. Moreover, HFBN(2, 7, 1) can also be scaled up-to 268 million of cores, which is enough number of cores for requiring a exa-scale system. This analysis considers only 500 simulation cycle to reduce the simulation execution time. On the other hand, other comparator networks show high congestion rate and requires large transfer time to run the same packet request probability (r). For example- with the same request probability, HFBN(2,6,1) can transfer about 388,620 packets and TTN(3,4,1) can transfer only 3,187 packets. On the other hand, 2DMesh can only send about 174 packets. Table A.1 shows the considered parameters for this analysis and Figure A.1 shows the uniform traffic analysis of HFBN(2,6,1). This graph also includes the traffic analysis of HFBN(2,5,1) for 1M cores and HFBN(2,4,1) for 65K Cores. On the other hand, Table A.2 and Table A.3 shows the parameters of traffic analysis for 1M and 65K cores respectively. Figure A.2, A.3, A.4 shows the traffic analysis with respect to uniform traffic, perfect shuffle traffic and bit-compliment traffic with 1M cores. Figure A.5, A.6 and A.7 shows the traffic analysis of 65K cores with uniform, perfect shuffle and bit-compliment traffic respectively.

Table A.1: Simulation Environment for Traffic Analysis

Parameter	Value	Units
Flow Control	Wormhole Switching	-
Channel Buffer Size	4	-
Simulation Cycle	500	Clock cycles
Virtual Channels	8	-
Router Pipeline Cycle	1	clocks
link Latency	1	clocks
Traffic Pattern	Uniform Traffic	-
Packet size	12	flits

Table A.2: Simulation Environment for Traffic Analysis (1M Cores)

Parameter	Value	Units
Cores	1,048,576	-
Flow Control	wormhole switching	-
Packet Size	6 (+ 6 Header flits)	flits
Channel Buffer Size	4	-
Simulation Cycle	5,000	Clock cycles
Virtual Channels	6	-
link latency	1	clocks
Router Pipeline Cycle	1	clocks

Table A.3: Simulation Environment for Traffic Analysis(65K Cores)

Parameter	Value	Units
Cores	65,536	-
Flow Control	wormhole switching	-
Packet Size	6 (+ 6 Header flits)	flits
Channel Buffer Size	2	-
Simulation Cycle	5,000	Clock cycles
Virtual Channels	4	-
link latency	1	clocks
Router Pipeline Cycle	1	clocks

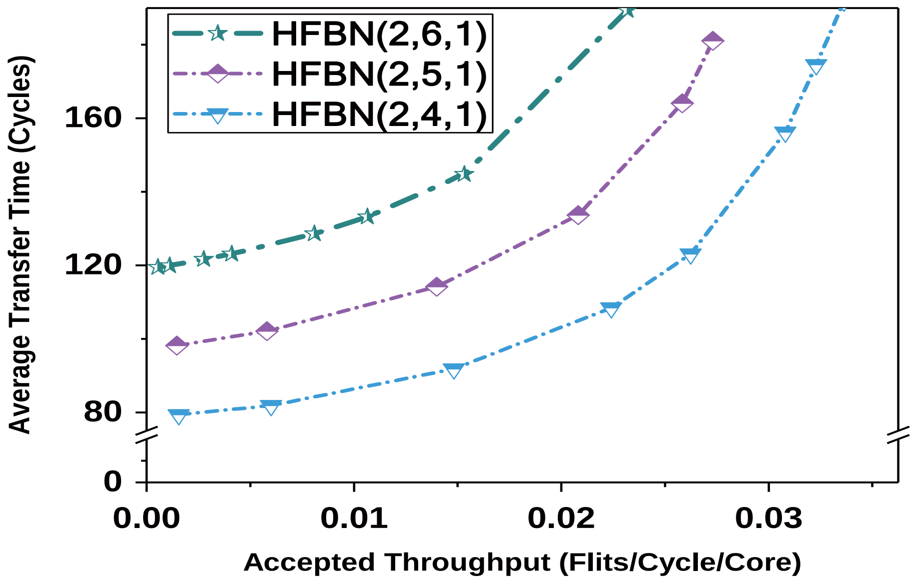


Figure A.1: Uniform Traffic Analysis with Various Network Size of HFBN

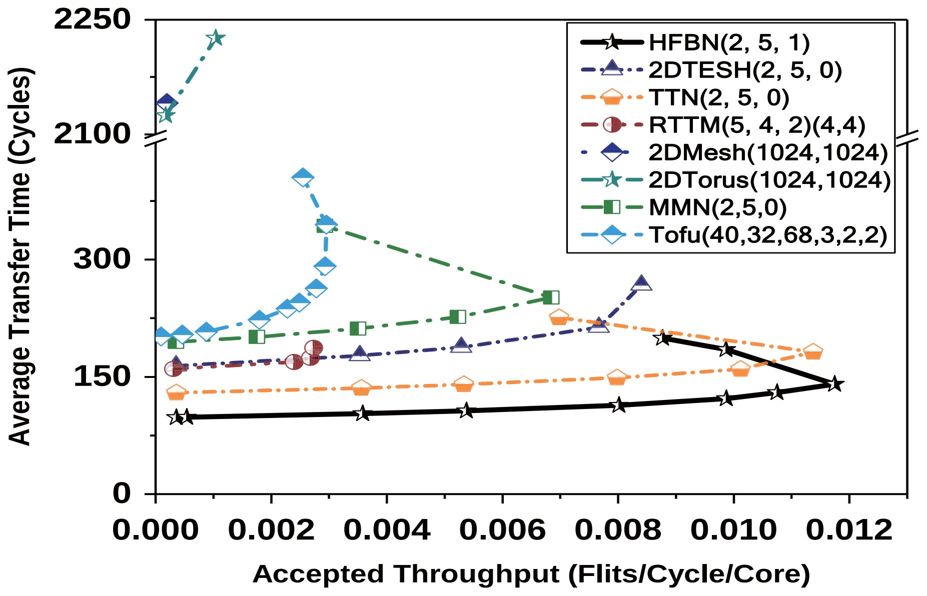


Figure A.2: Various networks with uniform traffic (1M Cores)

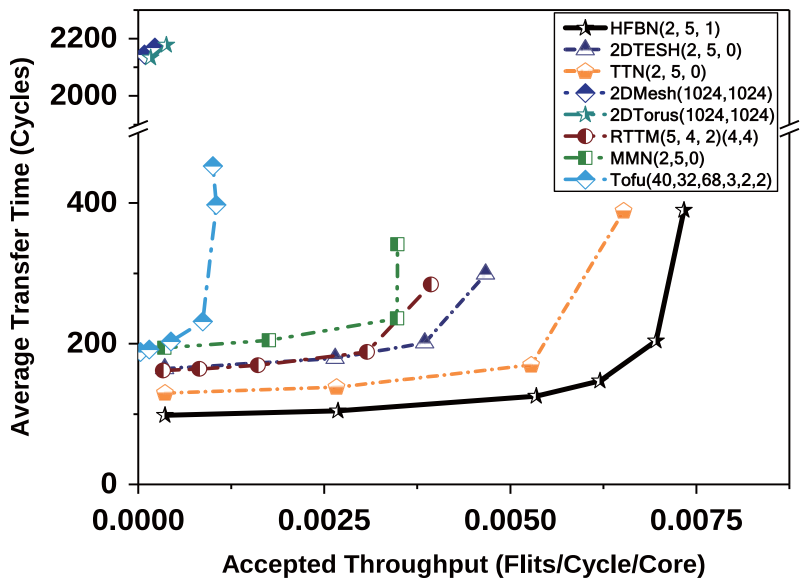


Figure A.3: Various networks with Perfect Shuffle traffic (1M Cores)

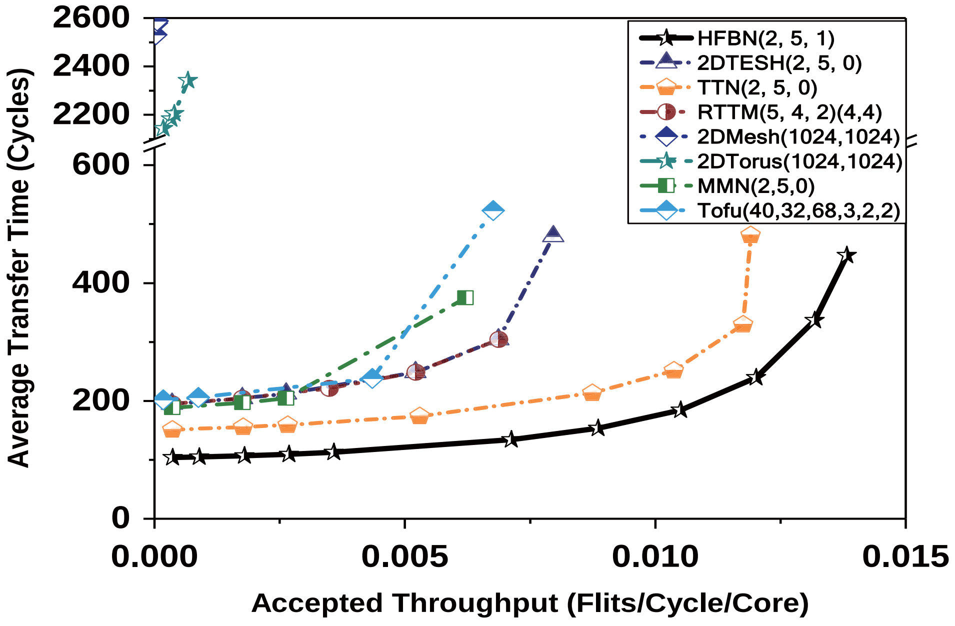


Figure A.4: Bit-compliment Traffic Analysis for various networks (1M cores)

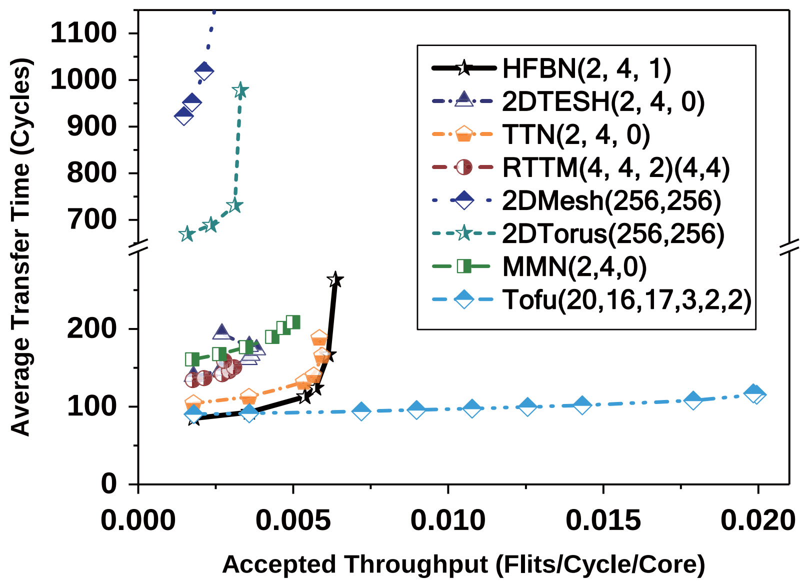


Figure A.5: Various networks with uniform traffic (65K Cores)

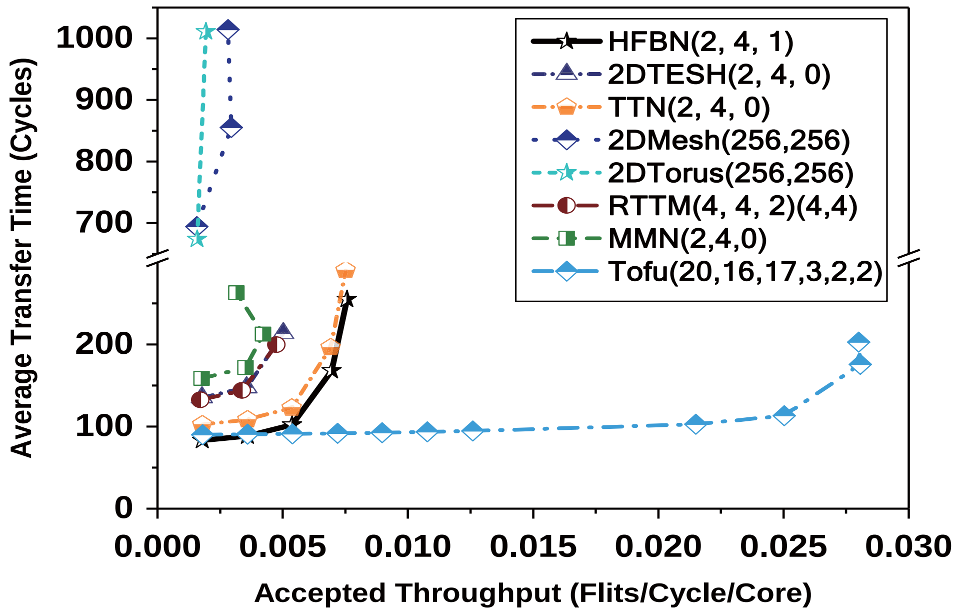


Figure A.6: Various networks with Perfect Shuffle Traffic (65K Cores)

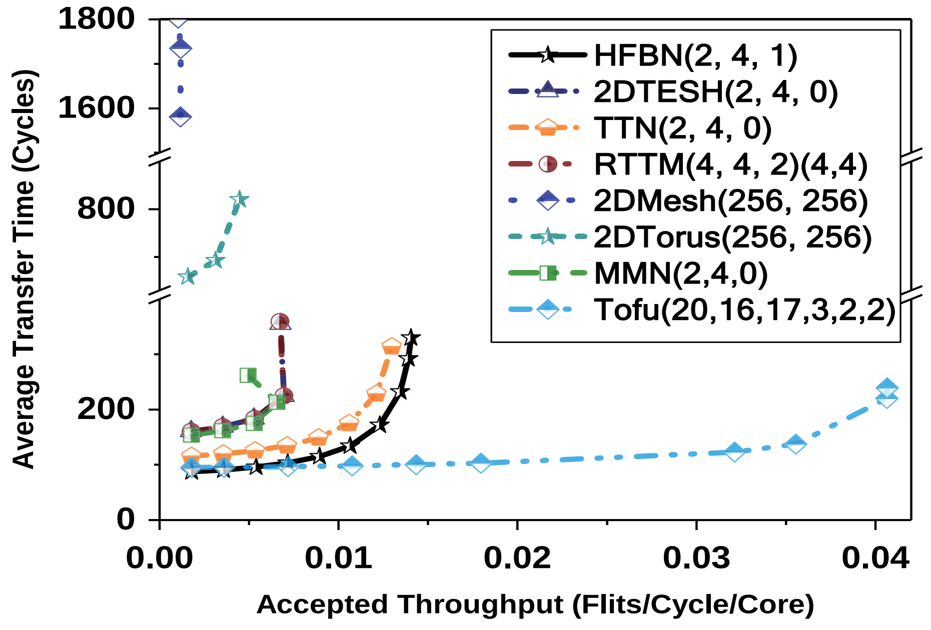


Figure A.7: Various networks with Bit-compliment Traffic (65K Cores)

Static Network Performance

Static network performance ensures the network capability without considering the packet movement. Hence, static network performance is useful in the initial choice of the network. Good network ensures low cost, low degree, low congestion, high connectivity and high fault-tolerant rate than the others. In this section, we compare the parameters, such as diameter, average distance and cost analysis. We consider up to the level-5 network performance for HFBN(2, 5, 1). Hence, we use SGI supercomputer with the openmp parallel programs run with the 6 core with 16 threads. Table A.4 shows the simulation environment for the static network performance. Static performance for Tofu has not been evaluated by the reference papers and hence, we skip the static analysis of Tofu for this section.

Table A.4: Static Simulation Enviornment

	SGI UV3000
OS	SUSE Linux Enterprise Server 12
CPU	Intel Xeon E5-465v3
Core	6 core
Number of Threads	16
Compiler	Intel C++ Compiler 17.0.1

Comparison of Static Performance of Various Networks

Node Degree

The node degree is defined as the maximum number of physical outgoing links from a core. Since each core of HFBN network has maximum 8 outgoing links, the degree of HFBN is 8. Constant node degree networks are easy to expand and the network interface cost remains unchanged with increasing network size. The I/O interface cost of a particular core is proportional to its degree. Table A.5 shows node degree for the various networks.

Table A.5: Node Degree of Various Networks

Parameter	2DMesh	2DTorus	TESH	TTN	RTTM	HFBN
Node Degree	4	4	4	6	4	8

5.2.1 Diameter Performance

The diameter ensures the maximum number of channels is required for a packet to be sent from each source core to destination core along its shortest path. However, static diameter doesn't consider the channel faults. Low diameter ensures low communication delay. Hence, the low diameter is preferable for any interconnection networks. Equation A.1 shows the diameter evaluation for HFBN(m, L, q) and Table A.6 shows the calculated formulation considering Equation A.1. On the other hand, Figure A.8 shows the diameter analysis of HFBN(2, L, 1) comparing with the various networks. This simulation ensures that diameter performance of HFBN(2, L, 1) is much better than various hierarchical networks (Like- TTN [23], TESH). In comparing the conventional networks, such as 2DMesh and 2DTorus, HFBN(2, 5, 1) shows much better results. Even HFBN(2, 5, 1) can achieve about 34.15% better than the TTN network. Diameter for HFBN can also be evaluated using the below equation-

For the HFBN, an upper bound for the diameter is given by-

$$Diameter = max(D_s + (\sum_{i=2}^{L}(D_{si} + D_i)) + D_d) \tag{A.1}$$

Here, where D_s = distance for highest level of outgoing core. D_{si} = distance for the next level of routing and D_i = distance for corresponding level of routing. D_d = distance from level-2 to destination core. Table A.4 shows the calculated formulation for HFBN network-

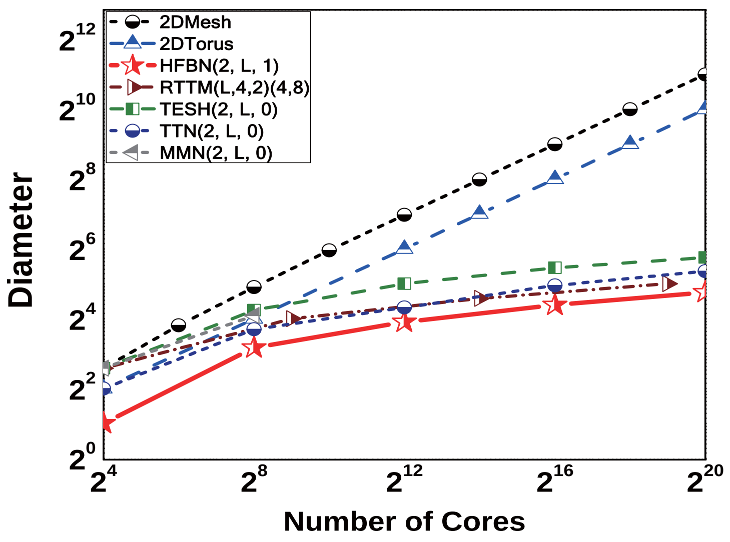


Figure A.8: Diameter Performance for Various Networks

Table A.6: Calculated Formulation of Diameter for HFBN

Parameter(m, L, q)	Result of D_s	Result for D_{si} and D_i	Result for D_d	Diameter of the corresponding level
HFBN(2, 1, 1)	2	$D_{si} = 0$, $D_i = 0$	0	2
HFBN(2, 2, 1)	2	for i = 2; D_{si}=0, D_i=5	2	9

HFBN(2, 3, 1)	2	for i = 2; D_{si}=1, D_i=5
		for i = 3; D_{si}=0, D_i=5
			2	15

HFBN(2, 4, 1)	2	for i = 2; D_{si}=1, D_i=5
		for i = 3; D_{si}=1, D_i=5
		for i = 4; D_{si}=0, D_i=5
			2	21

HFBN(2, 5, 1)	2	for i = 2; D_{si}=1, D_i=5
		for i = 3; D_{si}=1, D_i=5
		for i = 4; D_{si}=1, D_i=5
		for i = 5; D_{si}=0, D_i=5
			2	27

HFBN(2, 6, 1)	2	for i = 2; D_{si}=1, D_i=5
		for i = 3; D_{si}=1, D_i=5
		for i = 4; D_{si}=1, D_i=5
		for i = 5; D_{si}=2, D_i=5
		for i = 6; D_{si}=0, D_i=5
			2	34

5.2.2 Average Distance

Diameter analysis considers the routing for a single packet with a maximum path required to traverse along its shortest distance [53]. On the other hand, average distance considers the broadcasting of packets from each core to every other core. Hence, an average shorter path is more preferable over the low diameter. The average distance is the mean distance between each distinct pair of cores. The small average distance allows small communication latency. The average distance of graph G can be defined by Equation A.2, where n is the total number of cores in the network and d defined as the diameter between all distinct pair x and y. However, Figure A.9 shows the average distance of various networks, which ensures that HFBN is superior over TTN, 2DMesh, 2DTorus and TESH. Unfortunately, the average distance calculation for RTTM is wrong and hence, we ignored this network for this analysis.

$$\mu(\mathrm{G}) = \frac{\sum_{x,y \,\epsilon\, v} \mathrm{d(G;\, x,\, y)}}{\mathrm{n(n\ -\ 1)}} \tag{A.2}$$

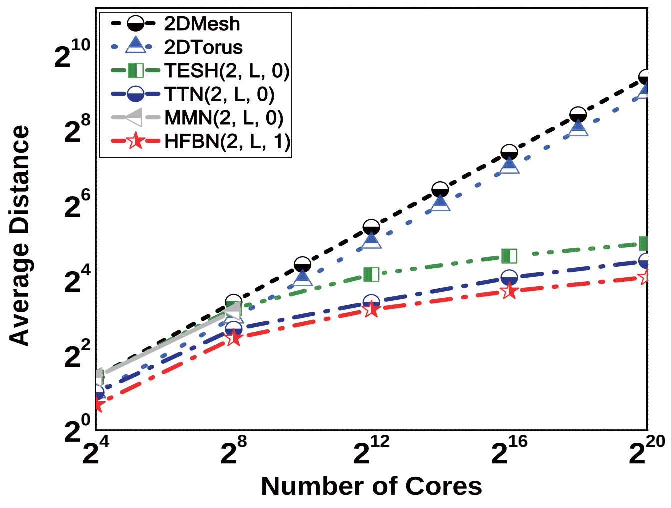


Figure A.9: Average Distance for Various Networks

Cost Analysis

Cost analysis is effective for interconnection networks due to its considerations in product of core degree and diameter. The cost and performance of network are inter-related due to inter-node distance, message traffic density and fault-tolerance. Node degree of a network can be defined as the maximum number of physical outgoing channels from a single core. The node degree of HFBN is 8. On the other hand, network radix can be defined by the number of channels for inter-router and number of cores are connected to a single router. Hence, the network radix for HFBN is 9 (8 links are used to connect other routers and single link will be used for connecting the single core). In contrast, this network is not fixed for connecting multiple cores from a single router. This feature allows the high network scalability for HFBN. Figure A.10 shows the cost analysis for HFBN(2, L, 1), which shows that this network outperformed the 2DTorus and even much better than the TTN network. On the other hand, RTTM and butterfly network shows the low cost performance due to its low node degree.

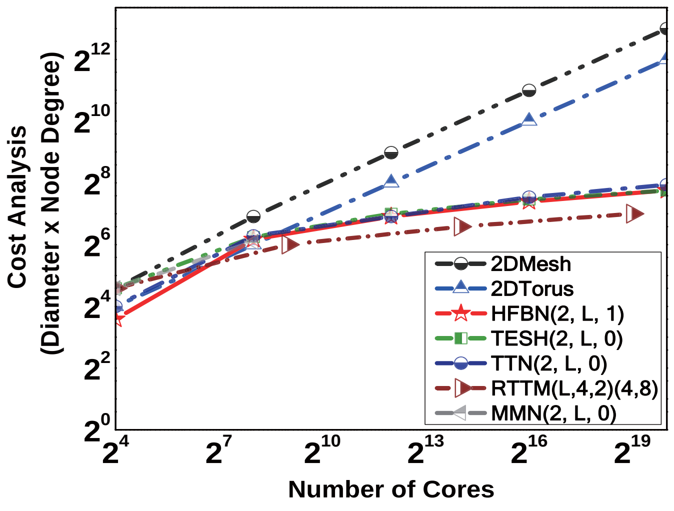


Figure A.10: Cost analysis for Various Networks

Topological Analysis

Packing Density

Network cost is defined as the product of network diameter and node degree. The network cost for HFBN has already been evaluated in the earlier studies and shows much better result than the 2DMesh, 2DTorus and even TTN networks. On the other hand, high packing density is one of the most desirable feature in VLSI, is defined as the ratio of the number of cores of a network to its cost [54]. The higher the packing density, the smaller chip area will be required for its VLSI layout. Equation A.3 shows the definition for the packing density. As the computational power for supercomputers are highly depends upon the multi-core structures, the chip area has a big impact on power usage. Figure A.11 shows the packing density for various networks. Its shows that HFBN network has a higher packing density than 2DMesh, 2DTorus and almost similar to TTN, TESH networks.

$$\text{Packing Density} = \frac{\text{Total Number of Cores}}{\text{Degree} \times \text{Diameter}} \tag{A.3}$$

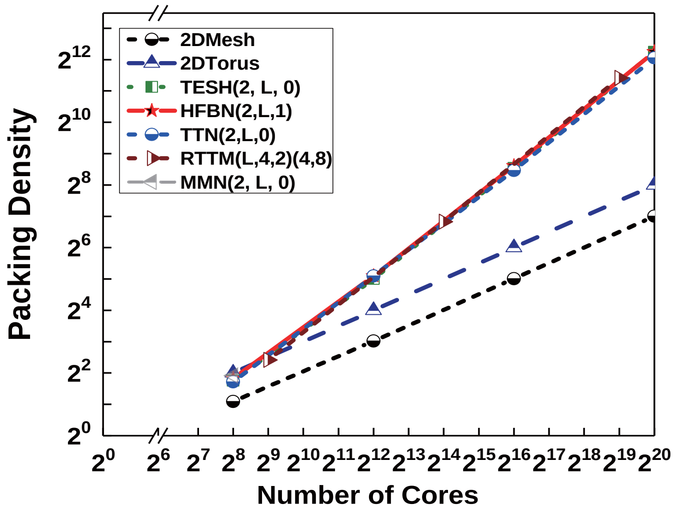


Figure A.11: Packing density for Various Networks

Message Traffic Density

The performance of a network for the message traffic can be evaluated by the average distance from one source core to other. An efficient network should have low message traffic density to reduce the traffic congestion and eventually should provide wide network bandwidth. The message traffic density is the ratio of multiplication between the total number of cores and its average distance to its total number of links [55]. Hence, message traffic density can be derived by the equation A.4.

$$\text{Message Traffic Density, } \rho = \frac{\text{dN}}{\text{E}} \tag{A.4}$$

Here, d is the average distance, N is the total number of cores and E is the total number of links. The total number of links for HFBN can be derived from the Equation A.5.

$$E = N_{BM} \times \text{inner } L_1 \text{ Links} + \sum_{i=2}^{N} N_{BM} \times \text{outer } L_i \text{ Links} \tag{A.5}$$

Here, N_{BM} is the number of basic module in current level, L_1 links considers for number of level-1 links and Li considers the number of i-th level links. Using those equations we have also compared the message traffic density with the other networks in Figure A.12, which shows that HFBN ensures the lowest message traffic density than the 2DTorus, 2DMesh, TTN, TESH networks. However, the estimated average distance for RTTM is not correct. Hence, we couldn't include the result for RTTM in this figure.

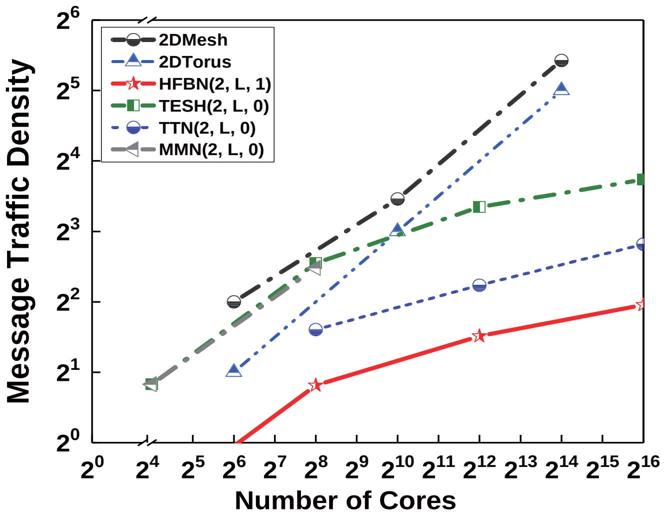


Figure A.12: Message Traffic density for Various Networks

Fault tolerance

Fault tolerance for networks directly related to network resiliency. In parallel computing environment fault tolerance is an important factor due to the common phenomenon of core or link failures. Fault tolerance for a graph has been defined as the maximum number of vertices that can be removed until the graph is still connected. Hence the fault tolerance for a graph is one less than its connectivity [56]. A network will be k-fault tolerant if it can sustain up to k number of link failures. In case of HFBN, the connectivity is lower than the node degree. Hence HFBN can tolerate up to (connectivity 1 = 5) five link failures at the BM level. Figure A.13 shows the total number of paths between each source-destination pair for the BM levels. This figure ensures that HFBN has the higher fault tolerance than 2DMesh, 2DTorus, TESH and TTN networks.

Cost Effectiveness factor

Speedup and efficiency are the usual measured parameters for the performance evolution of MPC systems. However, those parameters don't consider communication links. It is obvious for MPC system, the total system cost depends on the number of processor cores and with the number of communication links [56, 57]. The cost effectiveness factor (CEF) for HFBN has been defined by the Equation A.6. Let, CP is the cost for a processor core including its processing unit, control unit and memory unit. And, CL is the cost for the communication link. Hence ρ is defined as CL/CP. Figure A.14 shows the cost effectiveness of various networks ($\rho = 0.1$). Hence, HFBN is a good choice over the 2DMesh, 2DTorus as well as TESH network.

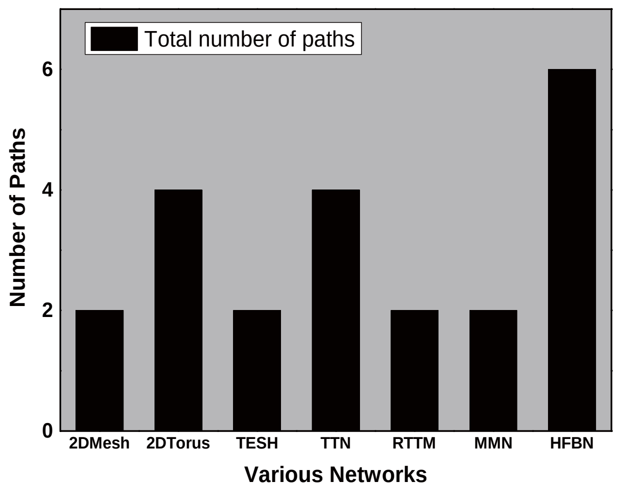


Figure A.13: On-chip connectivity for Various Networks

$$\text{CEF(N)} = \frac{1}{1 + \rho(\frac{\text{Total Number of communication links}}{\text{Total Number of Cores}})} \tag{A.6}$$

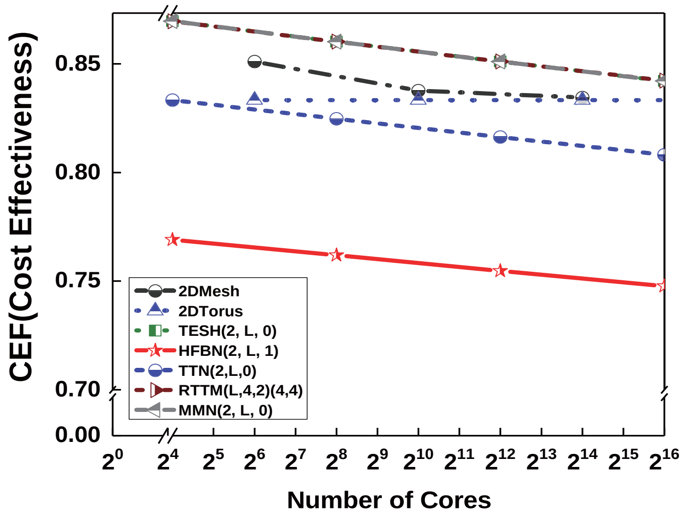


Figure A.14: CEF for Various Networks

Time Cost Effectiveness Factor

Time cost effectiveness factor (TCEF) parameter considers the time for the simulation seconds of a program [56, 57]. A faster solution is expected with the low cost effectiveness for the MPC systems. This factor helps to characterise the profitable use of the MPC systems. The TCEF for HFBN has been defined in equation A.8, where ρ is defined as CL/CP; same as cost-effectiveness. T_1 is the time to solve a single problem by a single processor core; we have assumed $\sigma = 1$, which is linear time penalty in T_p. T_p is the time required by p cores to solve a problem. Figure A.15 shows the TCEF for HFBN network, which is better than all the cost efficient networks.

$$\mathrm{g(p)} = \frac{\text{Total Number of communication links}}{\text{Total Number of Cores}} \tag{A.7}$$

$$TCEF(p, T_p) = \frac{1 + \sigma T_1^{\sigma-1}}{1 + \rho g(p) + \dfrac{\sigma}{p} T_p^{\sigma-1}} \tag{A.8}$$

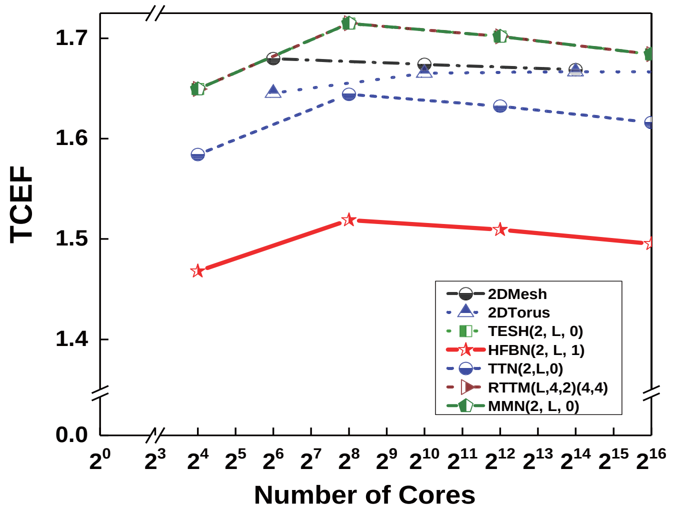


Figure A.15: TCEF for Various Networks

Appendix B

Simulator Verification

In our research, we have considered 2 simulator mainly for the performance evaluation and the analysis of the power estimation. In case of the power estimation, we have considered the Orion 2.0 power simulator [33]. This simulator is already been verified with the Intel 80 core [33]. On the other hand, considering the paper from DSENT power simulator, Orion 2.0(re-calibrated) simulator has a error rate of 47.5% in compared to SPICE simulation and about 50.36% error rate than the DSENT simulator in comparing the total router power [64]. However, the leakage power modeling is not accurate for DSENT simulator with different technology nodes, which leads us to consider the Orion 2.0 with default enviornment for this research. However, in this section, we would like verify the Orion 2.0 simulator setup same as the paper for obtaining the same result. Through the paper [33] has some of the important parameters are missing, we could able to obtain the same result as the paper considering the dynamic router power usage. Table B.1 shows those parameters (along with the given parameters as HVT, NVT, and LVT, we use (0.8 V, 0.2 GHz), (1.0 V, 1GHz), and (1.1 V, 3 GHz)) and Figure B.1 shows the paper results along with our evaluated results in Figure B.2. Our analysis showed that we may have only about 2% error rate in comparing with the referenced paper.

In the next part, we also like to verify our DCP traffic simulator. We have considered the performance analysis of Mesh network with the DOR routing [53] with the similar parameters. However, our simulator considers the wormhole flow control and referenced paper considers the virtual-channel flow control. Moreover, referenced paper considers the

Table B.1: Simulation Condition for Power Analysis

Parameter	Value	Units
Fabrication process	65/45	nm
number of Cores	1	Cores
Traffic pattern	uniform traffic	-
Message inject rate	0.28	flits/cycle/core
Simulation Cycle	20,000	-
Virtual Channels	2	-
Buffer Size	4	-

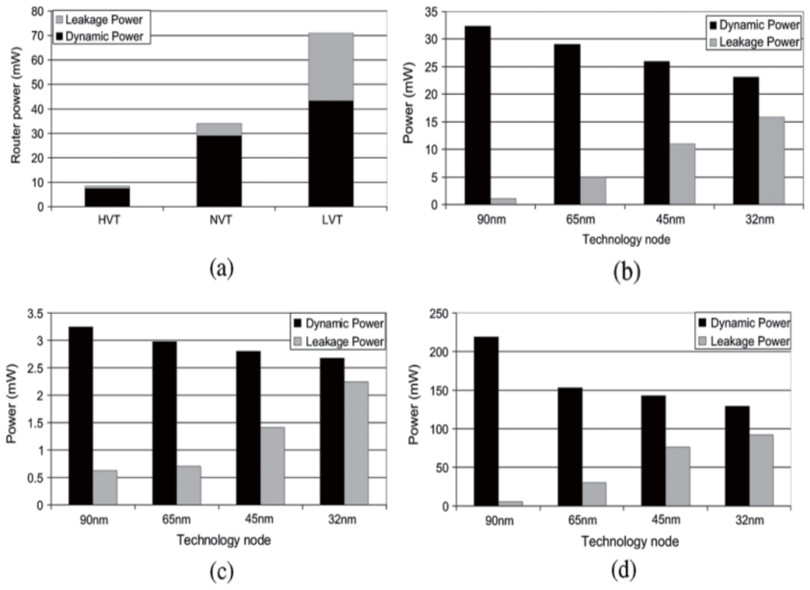


Figure B.1: Results of Orion 2.0 [33]

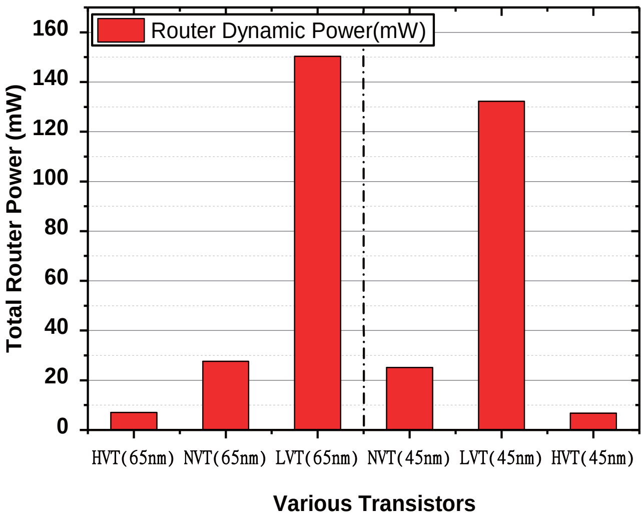


Figure B.2: Verification for Orion 2.0

per-hop latency of the routers is 3 cycles and our simulator considers 1 clock for router pipeline with 1 clock for the link traversal. Figure B.3 shows the DCP analysis for Mesh (8-ary 2-Mesh) from the referenced paper and Figure B.4 shows the evaluation analysis obtained from our simulator. In both cases of DOR routing, the zero load latency is almost similar. However, saturation rate showed in our simulator is much faster than the referenced paper. This difference leads to a about 39% difference at the saturation point.

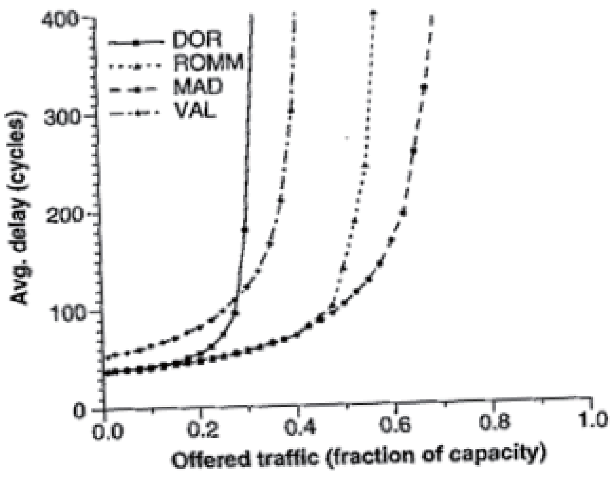


Figure B.3: Verification for DCP Simulator [53]

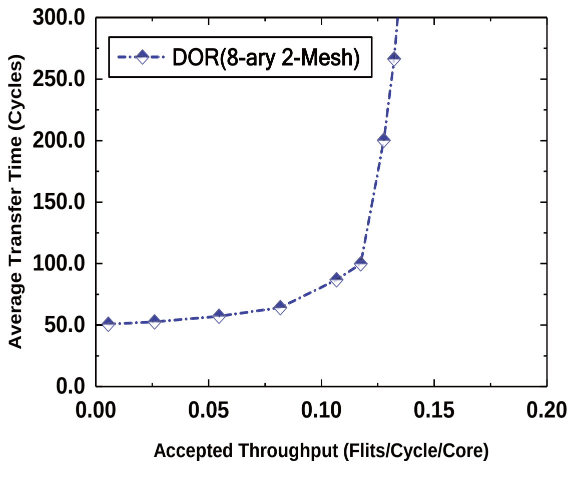


Figure B.4: Verification for DCP analysis on Mesh and Torus network

Appendix C

Appendix C is considered for the future analysis on HFBN. This chapter shows the evaluation method with the sample code to run the various evaluation of HFBN. Starting from the sample code of the network design with the "Higher Level Port Assignment for HFBN", then the sample code to obtain the "Static Performance Analysis" and finally, the sample code to obtain the "Dynamic Performance Analysis" of HFBN.

Higher Level Port Assignment for HFBN

The higher level port assignment for HFBN is required for the upper-level connection setup from the each basic module. The Algorithm 1 defined in Chapter 3 is also designed based on the below code implementation. This code requires input data as the value of m and q. And, the output will be the specific core number for the each higher-level connectivity. With the increase of paired connectivity(q), the number of out going port for each higher-level will be increased.

```
#define MX 256

/* defining the core struct with x, y position */
typedef struct{
    int x, y;
} yx;

int     m = 3;
int     BMAX;
int     Lmax;
int QMAX;
short   LV[MX][MX], HV[MX][MX], IN[MX][MX];

int ipow( int n ){
    int p=1;
    while( n-- )        p *= 2;
    return( p );
}


yx set_xy( int y, int x ){
    yx p;
    p.x = x;    p.y = y;
    return(p);
}

void setHVLQ( int y, int x, int pl, char phv, int connect ){
    LV[y][x] = pl;
    HV[y][x] = phv;
    IN[y][x] = connect;
}

 /* Initialising the pre-defined ports */
void initialize(int q){
    int x, y;
    yx  p;

    for( x=0; x <= BMAX; x++ ){
                for( y=0; y <= BMAX; y++ ){
                        LV[y][x] = 0;
                        HV[y][x] = '*';
                        IN[y][x] = 0;
                }
    }
    BMAX = ipow( m ) - 1;
    Lmax = ceil(2*(ipow( m ) - 1)/q) + 1;
    if (q < 2){
        setHVLQ(    0,    0, 2, 'H', 1 );
        setHVLQ(    0, BMAX, 2, 'V' , 1);
        setHVLQ( BMAX,    0,  3, 'V', 1 );
        setHVLQ( BMAX, BMAX, 3, 'H', 1 );
    }
    else {
                 setHVLQ(    0,    0, 2, 'H', 1 );
                        setHVLQ(    0, BMAX, 2, 'V', 1 );
                        setHVLQ( BMAX,    0,  2, 'V', 2 );
                        setHVLQ( BMAX, BMAX, 2, 'H', 2 );
    }
}

void print_portQ(){
    int x, y;

    for( y=BMAX; y >= 0; y-- ){
        for( x=0; x <= BMAX; x++ ){
         printf( "%3d%c_%d_", LV[y][x], HV[y][x], IN[y][x]);
        }
        printf( "\n" );
    }
}

/* check the previous level which is already assigned (pl is the
level number and phv is for the choice of vertical or Horizontal
selection and connect is to defined the current core number) */
yx port_levelQ( int pl, char phv , int connect){
    yx p;
    for( p.y=0; p.y <= BMAX; p.y++ ){
                for( p.x=0; p.x <= BMAX; p.x++ ){
                        if( pl == LV[p.y][p.x] && phv == HV[p.y][p.x]
                        && connect == IN[p.y][p.x]) return( p );
                }
    }
}

/* check the current level port is already assigned or not (pl is the
level number and phv is for the choice of vertical or Horizontal selection
and connect is to defined the current core number) */
bool availableQ( int pl, char phv , int connect){
    yx p;
    for( p.y=0; p.y <= BMAX; p.y++ ){
                for( p.x=0; p.x <= BMAX; p.x++ ){
                        if( pl == LV[p.y][p.x] && phv == HV[p.y][p.x]
                        && connect == IN[p.y][p.x]) return true;
                }
    }
    return false;
}

/* This function returns the mesh distance between the
 source and destination node */
int distance( yx src, yx dst ){
    yx  dist;
    dist.x = abs( src.x - dst.x );
    dist.y = abs( src.y - dst.y );
    return( dist.x + dist.y );
}


/* this function returns the y and x port position
of current level through $save\_ph$ and $save\_pv$*/
void SaveXY(int x,int y, yx* save_pv, yx* save_ph, yx lvmv, yx lvmh,
int dist, int *min_dist){

        yx  pv, ph;

        if( LV[0][x] == 0 ){
           pv = set_xy(0,x);     ph = set_xy(BMAX,x);
           dist = distance(pv, lvmh) + distance(ph, lvmv);
                if( dist < *min_dist ){
                        *save_pv = pv; *save_ph = ph; *min_dist = dist;
            }
           ph = set_xy(0,x);     pv = set_xy(BMAX,x);
           dist = distance(pv, lvmh) + distance(ph, lvmv);
                if( dist < *min_dist ){
                      *save_pv = pv;*save_ph = ph; *min_dist = dist;
             }
        }
        if( LV[y][0] == 0 ){
                pv=set_xy(y,0); ph=set_xy(y,BMAX);
                dist = distance(pv, lvmh) + distance(ph, lvmv);
                 if( dist < *min_dist ){
                                *save_pv = pv; *save_ph = ph; *min_dist = dist;
                 }
                ph=set_xy(y,0); pv=set_xy(y,BMAX);
                dist = distance(pv, lvmh) + distance(ph, lvmv);
                 if( dist < *min_dist ){
                                *save_pv = pv; *save_ph = ph; *min_dist = dist;
                 }
        }
}

/*  Sets the available port for upper levels*/
void HIGHERLevel(int m, int q){
           int  dist, min_dist;
           int x,y,i;
           int lv,max_connect;
           yx save_pv, save_ph, lvmv, lvmh;
           for( lv=2; lv <= Lmax; lv++ ){
               for ( i=1; i <= q; i++){
                        // check the already allocated max_connect
                        min_dist = 65536;
                        if (lv != 2 && i == 1){
                           lvmh = port_levelQ( lv-1, 'H', q );
                           lvmv = port_levelQ( lv-1, 'V', q );
                        }
                        else {
                                lvmh = port_levelQ( lv, 'H', i-1 );
                                lvmv = port_levelQ( lv, 'V', i-1 );
                        }
                        if (!availableQ(lv, 'V' ,  i)){
                                for( x=0, y = 0; x <= BMAX; x++, y++){
                                  SaveXY( x, y, &save_pv, &save_ph, lvmv,
                                  lvmh, dist, &min_dist);
                                }
                                setHVLQ( save_ph.y, save_ph.x, lv, 'H', i );
                                setHVLQ( save_pv.y, save_pv.x, lv, 'V', i );
                     }
               }
            }
}

int main( char *argc[], int argv ){
        int q = 4;
        initialize(q);
        HIGHERLevel(m,q);
        printf( "\nfinal\n" );
        print_portQ();
}
```

Static Performance Analysis

To run the static performance evaluation, we have considered the Openmp programming for the large-scale analysis. The Algorithm 2 in Chapter 2 also explains the static routing, which is considered for the HFBN. This code implementation is considered for HFBN(2,4,1) and the port assignment has been fixed by the Algorithm 1 in Chapter 3.

```
int difference(int dest, int src)
{
        return (dest - src);
}

int outlet_x(int *s, int *d, int l, int VH, int routedir)
{
        int outlet_nodex;
        if (l == 5 and VH == 1 and routedir == 0){return outlet_nodex = 0;}
        if (l == 5 and VH == 0 and routedir == 0){return outlet_nodex = 3;}
        if (l == 4 and VH == 1 and routedir == 0){return outlet_nodex = 3;}
        if (l == 4 and VH == 0 and routedir == 0){return outlet_nodex = 0;}
        if (l == 3 and VH == 1 and routedir == 0){return outlet_nodex = 0;}
        if (l == 3 and VH == 0 and routedir == 0){return outlet_nodex = 3;}
        if (l == 2 and VH == 1 and routedir == 0){return outlet_nodex = 3;}
        if (l == 2 and VH == 0 and routedir == 0){return outlet_nodex = 0;}

        if (l == 5 and VH == 1 and routedir == 1){return outlet_nodex = 0;}
        if (l == 5 and VH == 0 and routedir == 1){return outlet_nodex = 3;}
        if (l == 4 and VH == 1 and routedir == 1){return outlet_nodex = 3;}
        if (l == 4 and VH == 0 and routedir == 1){return outlet_nodex = 0;}
        if (l == 3 and VH == 1 and routedir == 1){return outlet_nodex = 0;}
        if (l == 3 and VH == 0 and routedir == 1){return outlet_nodex = 3;}
        if (l == 2 and VH == 1 and routedir == 1){return outlet_nodex = 3;}
        if (l == 2 and VH == 0 and routedir == 1){return outlet_nodex = 0;}

       return 0;
}

int outlet_y(int *s, int *d, int l, int VH, int routedir)
{
        int outlet_nodey;
        if (l == 5 and VH == 1 and routedir == 0){return outlet_nodey = 1;}
        if (l == 5 and VH == 0 and routedir == 0){return outlet_nodey = 1;}
        if (l == 4 and VH == 1 and routedir == 0){return outlet_nodey = 2;}
        if (l == 4 and VH == 0 and routedir == 0){return outlet_nodey = 2;}
        if (l == 3 and VH == 1 and routedir == 0){return outlet_nodey = 3;}
        if (l == 3 and VH == 0 and routedir == 0){return outlet_nodey = 3;}
        if (l == 2 and VH == 1 and routedir == 0){return outlet_nodey = 0;}
        if (l == 2 and VH == 0 and routedir == 0){return outlet_nodey = 0;}

        if (l == 5 and VH == 1 and routedir == 1){return outlet_nodey = 1;}
        if (l == 5 and VH == 0 and routedir == 1){return outlet_nodey = 1;}
        if (l == 4 and VH == 1 and routedir == 1){return outlet_nodey = 2;}
        if (l == 4 and VH == 0 and routedir == 1){return outlet_nodey = 2;}
        if (l == 3 and VH == 1 and routedir == 1){return outlet_nodey = 3;}
        if (l == 3 and VH == 0 and routedir == 1){return outlet_nodey = 3;}
        if (l == 2 and VH == 1 and routedir == 1){return outlet_nodey = 0;}
        if (l == 2 and VH == 0 and routedir == 1){return outlet_nodey = 0;}

        return 0;
}

int receiving_nodex(int *s, int *d, int l, int VH, int routedir)
{
        int receiving_nodex;
        if (l == 5 and VH == 1 and routedir == 0){return receiving_nodex = 0;}
        if (l == 5 and VH == 0 and routedir == 0){return receiving_nodex = 3;}
        if (l == 4 and VH == 1 and routedir == 0){return receiving_nodex = 3;}
        if (l == 4 and VH == 0 and routedir == 0){return receiving_nodex = 0;}
        if (l == 3 and VH == 1 and routedir == 0){return receiving_nodex = 0;}
        if (l == 3 and VH == 0 and routedir == 0){return receiving_nodex = 3;}
        if (l == 2 and VH == 1 and routedir == 0){return receiving_nodex = 3;}
        if (l == 2 and VH == 0 and routedir == 0){return receiving_nodex = 0;}

        if (l == 5 and VH == 1 and routedir == 1){return receiving_nodex = 0;}
        if (l == 5 and VH == 0 and routedir == 1){return receiving_nodex = 3;}
        if (l == 4 and VH == 1 and routedir == 1){return receiving_nodex = 3;}
        if (l == 4 and VH == 0 and routedir == 1){return receiving_nodex = 0;}
        if (l == 3 and VH == 1 and routedir == 1){return receiving_nodex = 0;}
        if (l == 3 and VH == 0 and routedir == 1){return receiving_nodex = 3;}
        if (l == 2 and VH == 1 and routedir == 1){return receiving_nodex = 3;}
        if (l == 2 and VH == 0 and routedir == 1){return receiving_nodex = 0;}

        return 0;
}

int receiving_nodey(int *s, int *d, int l, int VH, int routedir)
{
        int receiving_nodey;
        if (l == 5 and VH == 1 and routedir == 0){return receiving_nodey = 1;}
        if (l == 5 and VH == 0 and routedir == 0){return receiving_nodey = 1;}
        if (l == 4 and VH == 1 and routedir == 0){return receiving_nodey = 2;}
        if (l == 4 and VH == 0 and routedir == 0){return receiving_nodey = 2;}
        if (l == 3 and VH == 1 and routedir == 0){return receiving_nodey = 3;}
        if (l == 3 and VH == 0 and routedir == 0){return receiving_nodey = 3;}
        if (l == 2 and VH == 1 and routedir == 0){return receiving_nodey = 0;}
        if (l == 2 and VH == 0 and routedir == 0){return receiving_nodey = 0;}

        if (l == 5 and VH == 1 and routedir == 1){return receiving_nodey = 1;}
        if (l == 5 and VH == 0 and routedir == 1){return receiving_nodey = 1;}
        if (l == 4 and VH == 1 and routedir == 1){return receiving_nodey = 2;}
        if (l == 4 and VH == 0 and routedir == 1){return receiving_nodey = 2;}
        if (l == 3 and VH == 1 and routedir == 1){return receiving_nodey = 3;}
        if (l == 3 and VH == 0 and routedir == 1){return receiving_nodey = 3;}
        if (l == 2 and VH == 1 and routedir == 1){return receiving_nodey = 0;}
        if (l == 2 and VH == 0 and routedir == 1){return receiving_nodey = 0;}

       return 0;
}


long BM_routing(int sy, int sx, int dy, int dx)
{
        long diameter = 0;
        int movediry = 0;  //0 is for positive move and 1 is for negative move
        int movedirx = 0;

        int delx = dx - sx;
        int dely = dy - sy;

        if (dely > 0) {movediry = 0;}
        if (dely < 0) {movediry = 1;}
        //for Tori connection (0->2 or 1->3)
        if (movediry == 0 and dely == 2) {dely = dely - 1;}
        if (movediry == 0 and dely == -2) {dely = dely + 1;}
        if (movediry == 1 and dely == 2) {dely = dely - 1;}
        if (movediry == 1 and dely == -2) {dely = dely + 1;}
        //for Tori connection (0->3)
        if (movediry == 0 and dely > 2) {dely = dely - 4;}
        if (movediry == 1 and dely < -2) {dely = dely + 4;}

        if (delx > 0) {movedirx = 0;}
        if (delx < 0) {movedirx = 1;}
        //for Tori connection (0->2 or 1->3)
        if (movedirx == 0 and delx == 2) {delx = delx - 1;}
        if (movedirx == 0 and delx == -2) {delx = delx + 1;}
        if (movedirx == 1 and delx == 2) {delx = delx - 1;}
       if (movedirx == 1 and delx == -2) {delx = delx + 1;}
        //for Tori connection (0->3)
        if (movedirx == 0 and delx > 2) {delx = delx - 4;}
        if (movedirx == 1 and delx < -2) {delx = delx + 4;}



    while(dely != 0)
    {
                if (dely > 0) {dely = dely - 1; } //move the packet to +y direction
                if (dely < 0) {dely = dely + 1; } //move the packet to -y direction
                diameter++;
        }
        while(delx != 0)
        {
                if (delx > 0) {delx = delx - 1; } //move the packet to +x direction
                if (delx < 0) {delx = delx + 1; }//move the packet to -x direction
                diameter++;
        }
    return diameter;
```

```
}

//Defines the Shortest path routing...
int SP_Routing(int *s,int *d,int L,int i)
{
        int routedir = 0;

        if (((d[i] - s[i] + 4) % 4) > 4/2)
        {
                return routedir = 1;
        }
        else{
                return routedir = 0;
        }

}

long Routing( int s9,int s8, int s7, int s6, int s5, int s4,int s3,int s2, int s1,int s0,
int d9,int d8, int d7, int d6, int d5, int d4,int d3,int d2, int d1,int d0)
{
        int d[10] = { d0, d1, d2, d3, d4, d5, d6, d7, d8, d9};
        int s[10] = { s0, s1, s2, s3, s4, s5, s6, s7, s8, s9};
        int t[10] = {0};
        int outlet_nodex= 0;
        int outlet_nodey= 0;
        int diameter = 0;

        int routedir = 0; //0 is for positive and 1 is for negative move
                for (int i = 9; i>=2; i--)
                {
                        routedir = SP_Routing(s,d,floor((i)/2 + 1 ),i);

                    if (routedir == 0) { t[i] = (d[i] - s[i] + 4) % 4; }
                    else  { t[i] = 4 - ((d[i] - s[i] + 4) % 4); }

                        while(t[i] != 0)
                        {
                         if ((i % 2) == 0 ) {
                           outlet_nodex = outlet_x(s, d, floor((i)/2 + 1 ), 0, routedir);
                           outlet_nodey = outlet_y(s, d, floor((i)/2 + 1 ), 0, routedir);
                         }
                         else {
                          outlet_nodex = outlet_x(s, d, floor((i)/2 + 1 ), 1, routedir);
                          outlet_nodey = outlet_y(s, d, floor((i)/2 + 1 ), 1, routedir);
                         }
                         diameter += BM_routing(s[1],s[0], outlet_nodey,outlet_nodex);
                         if (routedir == 0) {
                           if (s[i] + 1 >= 4)
                                   s[i]= -4 + s[i] + 1;
                               else {s[i] = s[i] + 1;}
                         } // move the packet to the next BM
                         else {
                           if (s[i] - 1 < 0)
                                     s[i]= 4 + s[i] - 1;
                           else {s[i] = s[i] - 1;}
                          } // move the packet to the previous BM
                         diameter++;
                         if (t[i] > 0) t[i] = t[i] - 1;
                         if (t[i] < 0) t[i] = t[i] + 1;
                         if ((i % 2) == 0 ) {
                           s[0] = receiving_nodex(s,d, floor((i)/2 + 1 ), 0, routedir);
                           s[1] = receiving_nodey(s,d, floor((i)/2 + 1 ), 0, routedir);
                         }
                         else   {
                           s[0] = receiving_nodex(s,d, floor((i)/2 + 1 ), 1, routedir);
                           s[1] = receiving_nodey(s,d, floor((i)/2 + 1 ), 1, routedir);
                         }
                        }
                }

                diameter = diameter + BM_routing(s[1],s[0], d[1],d[0]);
        return diameter;
}


int main(void)
{
        long diameter = 0;
        long totalDiameter = 0;
        double averageDistance = 0;
        int maxDiameter = 0;
        int xMax = 4;
        int yMax = 4;
        int totalPair = 0;
        int l4sY,l4sX, l3sY, l3sX, sY, sX, sy, sx;
        int l4dY, l4dX, l3dY, l3dX, dY, dX, dy, dx;

#pragma omp parallel for collapse(14) private(dy,dx) reduction(+:totalDiameter)
    for(l4sY=0; l4sY<4; l4sY++)
    for(l4sX=0; l4sX<4; l4sX++)

     for(l3sY=0; l3sY<4; l3sY++)
     for(l3sX=0; l3sX<4; l3sX++)

     for(sY=0; sY<4; sY++)
     for(sX=0; sX<4; sX++)

     for(sy=0; sy<4; sy++)
     for(sx=0; sx<4; sx++){

          for(l4dY=0; l4dY<4; l4dY++)
          for(l4dX=0; l4dX<4; l4dX++)

          for(l3dY=0; l3dY<4; l3dY++)
          for(l3dX=0; l3dX<4; l3dX++)

          for(dY=0; dY<4; dY++)
          for(dX=0; dX<4; dX++)

          for(dy=0; dy<4; dy++)
          for(dx=0; dx<4; dx++){

              diameter = Routing(0, 0, l4sY, l4sX, l3sY, l3sX, sY, sX,
                     sy, sx, 0, 0, l4dY, l4dX, l3dY, l3dX, dY, dX, dy, dx);

              if(diameter >= maxDiameter){
                    maxDiameter = diameter;
              }
                      totalDiameter += diameter;
            }
         }

         averageDistance = totalDiameter / (double)(65536);
         averageDistance = averageDistance / (double)(65535);
        return 0;
}
```

Dynamic Communication Performance Analysis

Dynamic communication performance (DCP) is required for the network energy usage analysis. This analysis ensures the performance superiority of HFBN through the low zero-load latency and slower saturation rate. DCP analysis requires various important parameters for the deadlock-freeness as well as to maintain suitable network performance. Before discussing the running same codes, we could like to show the parameters which is required to setup the enviornment. Our simulator considers the NODEN as the number of cores, TIME is the total simulation cycles, P is the rate of packet generation (this parameter is required to change for the inrease of traffic loads in the network), C is the used for Hotspot traffic analysis and L is the packet length (This simulator considers 6 flits as the default and hence, L = 6, means a total of 12 flits as the packet length). We have considered the intel compiler with mcmodel(= medium) for the 1M and 16M core analysis and for 65K cores analysis, DCP simulation results are obtained from Visual C++ 2017 compilation.

```
    #define NODEN 1048576          /*  Number of PE             */
    #define LINEN     1024         /*  Number of PE in one direction  */
    #define LINKN        8         /*  Link Number of 1 PE      */
    #define VRCH         6         /*  Number of VC             */
    #define BUFN         4         /*  Capacity of Channel buffer         */
    #define P            2         /*  Probability of generating Packet  */
    #define TIME      5000         /*  Maximum Calculation time */
    #define SENDQN      16         /*  Capacity of sending buffer         */
    #define  MAXBUF     32
    #define SW\_MAX     10         /*  Maximum Number of Cross Bar Switch */
    #define RAND   077777          /*  RANDOM value                        */
    #define C         3277         /*   Communication Ratio in BM          */
    #define PMAX     32768         /*   Probability of generating Packet   */
    #define L            6         /*   Packet length                      */
    #define FIX          1         /*   Fixed Channel selection            */
```

Connectivity Setup

To analyze the DCP of HFBN, requires to follow the curtain steps for the code implementation. At first, we would like to explain the network configuration setup. Function linksetup() is considered for the network connection. We have considered HFBN(2,4,1) configuration to explain our link setup. This code requires the connectivity up to Level-4 network with paired connectivity is one. Hence, each BM will have only one Level-4, Level-3 and Level-2 vertical connection and one horizontal connection. Specific core number for the upper level connectivity has been determined by Algorithm 1(Chapter 3).

```
void linksetup(){

  // LINEN is the number of cores in X-direction in each BM = 2^m;
  // HFBN(2,4,1) has 4 cores in X-direction.
```

```
BMNODEN = LINEN * LINEN;

// This is the link setup for the 1st BM (0-16)
for(i=0;i<BMNODEN;i++){
        Nodelink[i][0] = i + LINEN;
        Nodelink[i][1] = i - LINEN;
        Nodelink[i][2] = i - 1;
        Nodelink[i][3] = i + 1;
        Nodelink[i][4] = i;
        Nodelink[i][5] = i;
        Nodelink[i][6] = i;
        Nodelink[i][7] = i;

   if(i >= BMNODEN - LINEN)
        Nodelink[i][0]=(i - BMNODEN + LINEN);
   if(i < LINEN)
        Nodelink[i][1]= i + BMNODEN - LINEN;
   if(i%LINEN == 0 )
        Nodelink[i][2]=i + (LINEN - 1);
   if(i%LINEN == LINEN - 1 )
        Nodelink[i][3]=i - (LINEN - 1);
   if (i < (BMNODEN / 2))
        Nodelink[i][4] = i + (LINEN * 2);
   if (i >= (BMNODEN / 2))
        Nodelink[i][4] = i - (LINEN * 2);
   if (i%LINEN < LINEN/2)
        Nodelink[i][5] = i + (LINEN / 2);
   if (i%LINEN >= LINEN / 2)
        Nodelink[i][5] = i - (LINEN / 2);
}


// THIS case is for the link setup for rest of the BMs
// NODEN is the total Core number in the network
for(i = BMNODEN; i< NODEN; i++){
          Nodelink[i][0] = Nodelink[i - BMNODEN][0] + BMNODEN;
          Nodelink[i][1] = Nodelink[i - BMNODEN][1] + BMNODEN;
          Nodelink[i][2] = Nodelink[i - BMNODEN][2] + BMNODEN;
          Nodelink[i][3] = Nodelink[i - BMNODEN][3] + BMNODEN;
          Nodelink[i][4] = Nodelink[i - BMNODEN][4] + BMNODEN;
          Nodelink[i][5] = Nodelink[i - BMNODEN][5] + BMNODEN;
          Nodelink[i][6] = i;
          Nodelink[i][7] = i;
}

// This  case for the horizontal links at L4_Horizontal_out
for (i = 8; i < BMNODEN * BMNODEN * BMNODEN * LINEN; i += BMNODEN) {
          Nodelink[i][6] = (i + BMNODEN * BMNODEN * BMNODEN) %
           (BMNODEN * BMNODEN * BMNODEN * LINEN);
          Nodelink[i][7] = (BMNODEN * BMNODEN * BMNODEN * LINEN +
          i - BMNODEN * BMNODEN * BMNODEN) % (BMNODEN * BMNODEN * BMNODEN * LINEN);
}
```

```
for (; i < NODEN; i += BMNODEN) {
        Nodelink[i][6] = Nodelink[i - BMNODEN * BMNODEN * BMNODEN * LINEN][6] +
        BMNODEN * BMNODEN * BMNODEN * LINEN;
        Nodelink[i][7] = Nodelink[i - BMNODEN * BMNODEN * BMNODEN * LINEN][7] +
        BMNODEN * BMNODEN * BMNODEN * LINEN;
}

// This case we are adding L3_Vertical out links
for(i = 12; i < BMNODEN * BMNODEN * BMNODEN; i+= BMNODEN){
        Nodelink[i][6] = (i + BMNODEN * BMNODEN * LINEN) %
        (BMNODEN * BMNODEN * BMNODEN);
        Nodelink[i][7] = (BMNODEN * BMNODEN * BMNODEN + i - BMNODEN * BMNODEN * LINEN)
        % (BMNODEN * BMNODEN * BMNODEN);
}
for (; i < NODEN; i += BMNODEN) {
        Nodelink[i][6] = Nodelink[i - BMNODEN * BMNODEN * BMNODEN][6] +
        BMNODEN * BMNODEN * BMNODEN;
        Nodelink[i][7] = Nodelink[i - BMNODEN * BMNODEN * BMNODEN][7] +
        BMNODEN * BMNODEN * BMNODEN;
}

// This case for the horizontal links at L3_Horizontal
for(i = 15; i < BMNODEN * BMNODEN * LINEN; i+= BMNODEN){
        Nodelink[i][6] = (i + BMNODEN * BMNODEN ) % (BMNODEN * BMNODEN * LINEN);
        Nodelink[i][7] = (BMNODEN * BMNODEN * LINEN + i - BMNODEN * BMNODEN )
         % (BMNODEN * BMNODEN * LINEN);
}
for(; i < NODEN; i+= BMNODEN){
        Nodelink[i][6] = Nodelink[i - BMNODEN * BMNODEN * LINEN][6] +
        BMNODEN * BMNODEN * LINEN;
        Nodelink[i][7] = Nodelink[i - BMNODEN * BMNODEN * LINEN][7] +
        BMNODEN * BMNODEN * LINEN;
}

// Level-2 Vertical Links connections
for(i = 3; i < BMNODEN * BMNODEN; i+= BMNODEN){
        Nodelink[i][6] = (i + BMNODEN * LINEN) % (BMNODEN * BMNODEN);
        Nodelink[i][7] = (BMNODEN * BMNODEN + i - BMNODEN * LINEN)  %
        (BMNODEN * BMNODEN);
}
for(; i < NODEN; i+= BMNODEN){
        Nodelink[i][6] = Nodelink[i - BMNODEN * BMNODEN][6] + BMNODEN * BMNODEN;
        Nodelink[i][7] = Nodelink[i - BMNODEN * BMNODEN][7] + BMNODEN * BMNODEN;
}


// Level-2 Horizontal Links connections
for(i = 0; i < (BMNODEN * LINEN); i+= BMNODEN){
        Nodelink[i][6] = (i + BMNODEN ) % (BMNODEN * LINEN);
        Nodelink[i][7] = (BMNODEN * LINEN + i - BMNODEN ) % (BMNODEN * LINEN);
}
for(; i < NODEN; i+= BMNODEN){
```

```
          Nodelink[i][6] = Nodelink[i - (BMNODEN * LINEN)][6] + (BMNODEN * LINEN);
          Nodelink[i][7] = Nodelink[i - (BMNODEN * LINEN)][7] + (BMNODEN * LINEN);
  }


  for(i=0;i<BMNODEN;i++)
    for(j = 0; j < LINKN; j++)
      for(k = 0; k < VRCH; k++)
        freechannel[i][j][k] = 0;

  for(i = BMNODEN; i< NODEN; i++)
    for(j = 0; j < LINKN; j++)
      for(k = 0; k < VRCH; k++)
        freechannel[i][j][k] = freechannel[i-BMNODEN][j][k];
}
```

DCP Routing

In DCP routing, each port number is mapped with the next possible core number (this is already considered in linksetup()) and hence, next receiving core number can be obtained from the corresponding port of the current source core. This case we have also considered the HFBN(2,4,1) network and hence, we need up to level-4 routing to complete the network simulation. Routing algorithm follows the highest level of routing first. Return value of Routing_HFBN() and BMroute() is the outgoing port number of the current source core.

```
int Routing_HFBN(int src, int dest){

  // Node number at the BM LEVEL...
  src0 = src % BMNODEN;
  dest0 = dest % BMNODEN;

  // NODE number at the LEVEL-2 NETWORK...
  src1 = (src % (BMNODEN*BMNODEN)) / BMNODEN;
  dest1 =  (dest % (BMNODEN*BMNODEN)) / BMNODEN;

  // NODE number at the LEVEL-3 NETWORK...
  src2 = ( src % (BMNODEN*BMNODEN*BMNODEN)) / (BMNODEN * BMNODEN);
  dest2 = (dest % (BMNODEN*BMNODEN*BMNODEN)) / (BMNODEN * BMNODEN);

  // NODE number at the LEVEL-4 NETWORK...
  src3 = src / (BMNODEN * BMNODEN * BMNODEN);
  dest3 = dest / (BMNODEN * BMNODEN * BMNODEN);


 // if level-4 network routing is required then we must make the level-4 routing.
 // The below case we make sure the level-4 routing is required...
      if (src3 != dest3) {
            // 4LV.V
            if ((src3 / LINEN) != (dest3 / LINEN)){
                  if (((LINEN + (dest3 / LINEN) - (src3 / LINEN)) % LINEN) <= LINEN / 2)
                    if (src0 == 11) return 6;
```

```
              else return BMroute(src0, 11);
            else
              if (src0 == 11) return 7;
                  else return BMroute(src0, 11);
            }

          // 4LV.H
          if ((src3 % LINEN) != (dest3 % LINEN)){
                if (((LINEN + (dest3 % LINEN) - (src3 % LINEN)) % LINEN) <= LINEN / 2)
                  if (src0 == 8) return 6;
                  else return BMroute(src0, 8);
                else
                  if (src0 == 8) return 7;
                  else return BMroute(src0, 8);
          }
}

// if level-3 network routing is required then we must make the level-3 routing.
// The below case we make sure the level-3 routing is required...
else if(src2 != dest2){
  // 3LV.V
  if((src2 / LINEN) != (dest2 / LINEN)){
    if(((LINEN +(dest2 / LINEN) - (src2 / LINEN)) % LINEN) <= LINEN / 2)
      if(src0 == 12) return 6;
      else return BMroute(src0, 12);
    else
      if(src0 == 12) return 7;
      else return BMroute(src0, 12);
      }

  // 3LV.H
  if((src2 % LINEN) != (dest2 % LINEN)){
    if(((LINEN +(dest2 % LINEN) - (src2 % LINEN)) % LINEN) <= LINEN / 2)
      if(src0 == 15) return 6;
      else return BMroute(src0, 15);
    else
      if(src0 == 15) return 7;
      else return BMroute(src0, 15);
      }
}

// if level-2 network routing is required then we must make the level-2 routing.
// The below case we make sure the level-2 routing is required...
else if(src1 != dest1){
  // 2LV.V
  if((src1 / LINEN) != (dest1 / LINEN)){
    if((( LINEN +(dest1 / LINEN)-(src1 / LINEN)) % LINEN) <= LINEN / 2)
      if(src0 == 3) return 6;
      else return BMroute(src0, 3);
    else
      if(src0 == 3) return 7;
      else return BMroute(src0, 3);
```

```
        }

    //  2LV.H
    if((src1 % LINEN) != (dest1 % LINEN)){
      if(((LINEN +(dest1 % LINEN)-(src1 % LINEN)) % LINEN) <= LINEN / 2)
        if(src0 == 0) return 6;
        else return BMroute(src0, 0);
      else
        if(src0 == 0) return 7;
        else return BMroute(src0, 0);
        }
  }
  // if level-1 network routing is required then we must make the level-1 routing.
  // The below case we make sure the level-1 routing is required...
   else return BMroute(src0, dest0);

}

// BMroute returns the outgoing port number at the on-chip level
int BMroute(int bmsrc, int bmdest) {

        src = bmsrc % BMNODEN;
        dest = bmdest % BMNODEN;

        // Src and dest are same....
        if (src == dest)
                return LINKN; // LINKN = NODE DEGREE of HFBN(8)
        else if (((src / LINEN) == 2 && (dest / LINEN) == 0) ||
        ((src / LINEN) == 3 && (dest / LINEN) == 1))
                return 4;
        else if ((src / LINEN) == 3 && (dest / LINEN) == 0)
                return 0;
        else if ((src / LINEN) == 0 && (dest / LINEN) == 3)
                return 1;
        else if (((src / LINEN) == 0 && (dest / LINEN) == 2)
        || ((src / LINEN) == 1 && (dest / LINEN) == 3))
                return 4;
        else if (src / LINEN > dest / LINEN)
                return 1;
        else if (src / LINEN < dest / LINEN)
                return 0;


        else if (((src % LINEN) == 2 && (dest % LINEN) == 0)
        || ((src % LINEN) == 3 && (dest % LINEN) == 1))
                return 5;
        else if ((src % LINEN) == 3 && (dest % LINEN) == 0)
                return 3;
        else if ((src % LINEN) == 0 && (dest % LINEN) == 3)
                return 2;
        else if (((src % LINEN) == 0 && (dest % LINEN) == 2)
        || ((src % LINEN) == 1 && (dest % LINEN) == 3))
```

```
                return 5;
        else if (src % LINEN > dest % LINEN)
                return 2;
        else if (src % LINEN < dest % LINEN)
                return 3;
        else { printf("Error at BM routing....\n"); }
}
```

Selection of the Virtual Channel

Selecting the virtual channel for the deadlock-free routing is the most important part for the hierarchical network because hierarchical network maintains various network configuration at different level of hierarchy. Priority_outbuffer() is considered for switching between the virtual channels for the deadlock-freeness. For example- the case when a packet is generated in the source node destined for other destination node, case 1 (1st flit is generated) is considered for the Source BM (case 0 is for the no flit is available at the source node). Then, the case 2 is considered for outer BM routing. If the wrap-around route is taken then the case 12 is considered. Case 22 is after the current level horizontal or vertical routing is completed. Finally, case 3 is for the Destination BM routing. If the 1st flit is available at the current node and then, inputbuffer() is responsible for updating the packet ID number depending on the current source and routing condition. This function updates the packet ID through case 1 clause (this case is considered when the 1st flit is arrived at the current source node).

```
// k is the selected virtual channel number, i is for the current node number
// j is the link number, id_temp is the defined as the packet ID.
int priority_outbuffer(int k, int i, int j, int id_tmp) {
        int m, id;

        m = (k / 2) * 2;
        id = id_tmp / 10000000;

        if (j == LINKN)
                return k;

        if (FIX || (freechannel[i][j][VRCH - 1] == 0))
                k = 0;

        if (k == 0) {
                switch (id) {
                case 0:
                        return 0 + m; // returns even-numbered virtual channel
                        break;
                case 1:
                        return 0 + m;
                        break;
                case 2:
                        if (j >= 6) return 0 + m;
                        else return 0 + m;
                        break;
                case 12:
                        if (j >= 6) return 1 + m; // returns odd-numbered virtual channel
                        else return 0 + m;
                        break;
                case 22:
                        return 0 + m;
                        break;
                case 3:
                        return 1 + m; // returns odd-numbered virtual channel
                }
        }
        else {

                if (id == 0)
                        return k;
                else {
                        m = k;
                        while (freechannel[i][j][m] == 0)
                                m = (m + 1) % VRCH;
                        return m;
                }
        }
}
// time is to track down the simulation cycle time
void inputbuffer(int time) {

  // ** Some Omitted Code which are not required for Virtual Channel Selection...
  case 1:

   // if nextroute return 6/7 then it means its a off-chip routing...
   // add +1 to move to a new BM and this is the first UPPER_LEVEL BM MOVE
   // As the initial id for BM level is 10000000 so we add 10000000
   // make 20000000 as the second BM ID...
   if ((nextroute(i, lines[i].inputbuf_dest[j][k]) >= 6)
   && (lines[i].inputbuf_id_next[j][k] / 10000000 == 1)) {
                lines[i].inputbuf_id_next[j][k] += 10000000; //ID +1
   }

   // case 2, case when we are making the move from out of second BM (same level V/H move)
   // So we will add +10 to send to a new BM id (same level V/H move)
   // then in packests id become +12 this will be the torus route
   // +22 for end of Vertical or Horizontal routing.
                if (((nextroute(i, lines[i].inputbuf_dest[j][k]) == 6)
                && (Nodelink[i][nextroute(i, lines[i].inputbuf_dest[j][k])] < i))
                || ((nextroute(i, lines[i].inputbuf_dest[j][k]) == 7)
                && (Nodelink[i][nextroute(i, lines[i].inputbuf_dest[j][k])] > i))) {

                  switch (lines[i].inputbuf_id_next[j][k] / 10000000) {
                        case  2:
                                lines[i].inputbuf_id_next[j][k] += 100000000; //+10
                            break;
                        case 12:
                                printf("error!!!! source=%d  destination=%d %d %d %d\n",
                                i, lines[i].inputbuf_dest[j][k],
                                nextroute(i, lines[i].inputbuf_dest[j][k]),
                                lines[i].inputbuf_id_next[j][k],
                                lines[i].inputbuf_dest_next[j][k]);
                                break;
                        case 22:
                                lines[i].inputbuf_id_next[j][k] -= 100000000; //-10
                        }
                }


// Change the ID number if current BM is not the destination BM
        if ((nextroute(i, lines[i].inputbuf_dest[j][k]) < 6) &&
        ((i / 16) != (lines[i].inputbuf_dest[j][k] / 16))) {
                switch (i % 16) {
                  case  0:
                  case  3:
                  case 12:
                  case 15:
                  case 8:
                  case 11:
                  case 4:
                  case 7:
                        if (lines[i].inputbuf_id_next[j][k] / 10000000 == 2)
                                lines[i].inputbuf_id_next[j][k] += 200000000; // +20
                        if (lines[i].inputbuf_id_next[j][k] / 10000000 == 12)
                                lines[i].inputbuf_id_next[j][k] += 100000000; // +10
                        break;
                 case  2:
                    if (((i%16384)/4096) == ((lines[i].inputbuf_dest[j][k]%16384)/4096)
                          && ((i % 16 == 2))) {
                           if (lines[i].inputbuf_id_next[j][k] / 10000000 == 2)
                                lines[i].inputbuf_id_next[j][k] += 200000000; // +20
                           if (lines[i].inputbuf_id_next[j][k] / 10000000 == 12)
                                lines[i].inputbuf_id_next[j][k] += 100000000; // +10
                         }
                 case 13:
                    if (((i%1024)/256) == ((lines[i].inputbuf_dest[j][k]%1024)/256)
                          && ((i % 16 == 2) || (i % 16 == 13))) {
                           if (lines[i].inputbuf_id_next[j][k] / 10000000 == 2)
                                  lines[i].inputbuf_id_next[j][k] += 200000000; // +20
                           if (lines[i].inputbuf_id_next[j][k] / 10000000 == 12)
                                  lines[i].inputbuf_id_next[j][k] += 100000000; // +10
                           }
                          break;
                        }
                 }

                        if ((nextroute(i, lines[i].inputbuf_dest[j][k]) >= 6)
                        && (lines[i].inputbuf_id_next[j][k] / 10000000 == 22))
```

```
                                    lines[i].inputbuf_id_next[j][k] -= 200000000;

                          if ((nextroute(i, lines[i].inputbuf_dest[j][k]) < 6)
                          && ((i / 16) == (lines[i].inputbuf_dest[j][k] / 16))) {
                                    if (lines[i].inputbuf_id_next[j][k] / 10000000 == 12)
                                              lines[i].inputbuf_id_next[j][k] -= 100000000;
                                    if (lines[i].inputbuf_id_next[j][k] / 10000000 == 2)
                                              lines[i].inputbuf_id_next[j][k] += 10000000;
                          }
                          break;


              // ** Some Omitted Code which are not required for Virtual Channel Selection...
}
```

Electrical Power Analysis

We have used the Orion 2.0 power simulator [33] along with the garnet 1.0 [34] for running the data driven power analysis. Electrical module is considered up to inter-chip level. Hence, up to 256 cores power usage has be obtained from single electrical power analysis. To obtain the total electrical power usage requires to multiply the number inter-chip module with the obtained single electrical power usage from the Orion power analysis. Here, we have showed the simulation parameters for power analysis and the sample code for HFBN(2,2,1) a single inter-chip module. The routing for this inter-chip analysis is considered with the default "Table-based" routing of the garnet 1.0 simulator.

```
// Running simulation parameters for garnet 1.0
./build/ALPHA_Network_test/gem5.debug configs/example/ruby_network_test.py
--num-cpus=256 --num-dirs=256 --topology=HFBN --mesh-rows=4  --sim-cycles=5000
--injectionrate=0.00179 --synthetic=0 --garnet-network=fixed --cpu-clock=1GHz
--sys-clock=1GHz

class HFBN(SimpleTopology):
    description='HFBN'

        def __init__(self, controllers):
        self.nodes = controllers

        def makeTopology(self, options, network, IntLink, ExtLink, Router):
        nodes = self.nodes

        num_routers = options.num_cpus
        num_rows = options.mesh_rows

        # There must be an evenly divisible number of cntrls to routers
        # Also, obviously the number or rows must be <= the number of routers
        cntrls_per_router, remainder = divmod(len(nodes), num_routers)
        assert(num_rows <= num_routers)
        num_columns = int(num_routers / num_rows)
        assert(num_columns * num_rows == num_routers)
    # Create the routers in the torus
    routers = [Router(router_id=i) for i in range(num_routers)]
    network.routers = routers

    # link counter to set unique link ids
    link_count = 0

    # Add all but the remainder nodes to the list of nodes to be uniformly
    # distributed across the network.
    network_nodes = []
    remainder_nodes = []
    for node_index in xrange(len(nodes)):
        if node_index < (len(nodes) - remainder):
            network_nodes.append(nodes[node_index])
        else:
            remainder_nodes.append(nodes[node_index])

    # Connect each node to the appropriate router
    ext_links = []
    for (i, n) in enumerate(network_nodes):
        cntrl_level, router_id = divmod(i, num_routers)
        assert(cntrl_level < cntrls_per_router)
        ext_links.append(ExtLink(link_id=link_count, ext_node=n,
                                 int_node=routers[router_id]))
        link_count += 1

    # Connect the remainding nodes to router 0.  These should only be
    # DMA nodes.
    for (i, node) in enumerate(remainder_nodes):
        assert(node.type == 'DMA_Controller')
        assert(i < remainder)
        ext_links.append(ExtLink(link_id=link_count, ext_node=node,
                                 int_node=routers[0]))
        link_count += 1

    network.ext_links = ext_links

    # Create the torus links.  First row (east-west) links then column
    # (north-south) links
    # column links are given higher weights to implement XY routing
    int_links = []

    BM_Col = num_rows
    BM_NUM = num_columns/num_rows
    TOTAL_Node =  num_columns * num_rows

    ################################################################
    ########         HFBN(2,1,X) intra-chip connection
    ################################################################
    # routing for X direction
    for Current_BM in xrange(BM_NUM):
```

```
    for row in xrange(num_rows):
        for col in xrange(BM_Col):
            west_id = col + (row * BM_Col) + (Current_BM * BM_NUM)
            for subcol in range(col, BM_Col-1):
                if (subcol + 1 < BM_Col):
                 east_id = (subcol + 1) + (row * BM_Col) + (Current_BM * BM_NUM)
                else:
                 east_id = (row * BM_Col) + (Current_BM * BM_NUM)
                int_links.append(IntLink(link_id=link_count,
                                         node_a=routers[east_id],
                                         node_b=routers[west_id],
                                         weight=1))
                link_count += 1



# routing for Y direction
for Current_BM in xrange(BM_NUM):
    for col in xrange(BM_Col):
        for row in xrange(num_rows):
            north_id = col + (row * BM_Col) + (Current_BM * BM_NUM)
            for subrow in range(row, num_rows-1):
                if (subrow + 1 < num_rows):
                 south_id = col + ((subrow + 1) * BM_Col) + (Current_BM * BM_NUM)
                else:
                 south_id = col + (Current_BM * BM_NUM)
                int_links.append(IntLink(link_id=link_count,
                                         node_a=routers[north_id],
                                         node_b=routers[south_id],
                                         weight=2))
                link_count += 1

################################################################
########       HFBN(2,2,1) Level2(off-chip) connection
################################################################
# routing for Level-2 Vertical direction
for col in xrange(3, num_columns,BM_NUM):
    for row in xrange(num_rows):
        north_id = col + (row * num_columns)
        if (row + 1 < num_rows):
            south_id = (col + ((row + 1) * BM_NUM * num_rows)) % TOTAL_Node
        else:
            south_id = (col) % TOTAL_Node
        int_links.append(IntLink(link_id=link_count,
                                 node_a=routers[north_id],
                                 node_b=routers[south_id],
                                 weight=2))
        link_count += 1

# routing for Level-2 Horizontal direction
for col in xrange(0, TOTAL_Node, num_columns):
    for row in xrange(num_rows):
            north_id = col + (row * BM_NUM)
            if (row + 1 < num_rows):
                south_id = (col + ((row + 1) * BM_NUM )) % TOTAL_Node
            else:
                south_id = (col) % TOTAL_Node
            int_links.append(IntLink(link_id=link_count,
                                    node_a=routers[north_id],
                                    node_b=routers[south_id],
                                    weight=2))
            link_count += 1

        network.int_links = int_links
```

Acknowledgments

Life goes on with lots of sorrows and sufferings to achieve the certain moment of happiness. Those sorrows and sufferings can be overcome from the interaction with a colleague, friend or a family member through their encouragement of passing difficult times, clarifying my understanding, improving in day to day work and supporting me throughout the research tenure. I would like to express my deepest appreciation to all the people, who had contributed in some way to complete this research and also like to point out certain individuals to whom my sorrows and sufferings mitigated immensely.

First of all, I would like to express my deepest acknowledgement to my supervisor, Professor Yasushi Inoguchi (School of Information Science, Japan Advanced Institute of Science and Technology) for his kind and honest support and effective advise during my research period. His guidance helped me to find the correct route of my research. His constant inspiration and encouragement support me to reach the final goal of my research.

Secondly, I like to express my sincere thanks to Professor Yasuyuki Miura (Department of Information Science, Shonan Institute of Technology) for the evaluation of dynamic communication performance and Assistant Professor M.M. Hafizur Rahman (Dept. of CN, CCSIT, King Faisal University, Saudi Arabia) for restructuring this thesis, helping me as the co-author of my journal paper and providing uninterrupted encouragement.

Then, I like to express my gratitude to second supervisor Associate Professor Kiyofumi Tanaka (School of Information Science, Japan Advanced Institute of Science and Technology) for his cordial advices during my research and forcing me to obtain the desired results for the next generation supercomputing with hierarchical interconnection networks. I also wish to express my honour to my sub-theme supervisor Professor Ryuhei Uehara (School of Information Science, JAIST) for allowing me to explore the puzzle problems.

I also like to continue my gratitude to all the committee member of my Ph.D. defence; Associate Professor Kiyofumi Tanaka, Professor Mineo Kaneko (School of Information Science, Japan Advanced Institute of Science and Technology), Professor Yasuyuki Miura and Associate Professor Yuto Lim (School of Information Science, Japan Advanced Institute of Science and Technology) for their kind advices to improve this dissertation.

I am eagerly acknowledging for granting me the ”2015-2018 Doctoral Research Fund, 2016 KDDI Scholarship and 2015 Monbukagakusho Honours Scholarship” which makes my study and life feasible in Japan. And also like to acknowledge ”Research Grants for JAIST Students” which allows me to present my research work at international conferences.

Finally, I like to give my sincere acknowledgments to my family members for their huge support and patience; to my father, my mother and also to my spouse.