Three-Dimensional Network-on-Chip Architectures for Cycle Accurate Full-System Simulator

A
Project Report
Submitted for the partial fulfillment
of M.S. degree
in
COMPUTER SCIENCE

by

Akash Agarwal
(OSU ID: 933-471-097)

Under the Supervision of
Dr. Lizhong Chen

Department of Electrical Engineering and Computer Science (EECS)
Oregon State University
Corvallis, OR

June, 2020

CONTENTS

ABSTRACT

High Performance Computing can find ubiquitous applications in the industry. HPC-applications are specifically designed to take advantage of the parallel nature of the computing systems which is often enabled by Multi-core/Many-core architectures. With this advent of Multi-processors in the mainstream systems, inter-core communication has been one of the major challenges as it affects the full-system performance, thus making Network-on-Chip, which connects different components, an integral part of chip design.

Since the On-Chip Network topology determines the physical layout and how connections are laid-out between the nodes, it has a profound impact on overall network-performance. Recently, there has been a surge in the number of articles proposing 3D mesh topologies for Systems research. This Master's report reflects the motivation and efforts needed to implement 3D topology models (3D mesh and 3D-stacked mesh) in the Gem5 Simulator to serve as a foundation in simulating all future 3D NoC related researches. Gem5 is a Cycle-accurate full-system simulator highly used in Computer Architecture research.

This project solely focuses on enabling 3D mesh Network simulations on GARNET which is a cycle-accurate interconnection network model inside Gem5 simulator and primarily consists of 2D network designs only. The report also discusses a very sound approach of determining the position of TSV links (vertical links) in the 3D stacked mesh NoC design.

LIST OF FIGURES

1. INTRODUCTION

1.1. Multi-Core Era

Till the end of twentieth century, Computer Architecture research had been pre-dominantly focused on powering single-core processors, exploiting the relationship between Moore’s law and Dennard’s Scaling law. This relationship cannot be understated. Moore observed that the number of transistors in a dense integrated circuit can double approximately every two years. The Dennard scaling related to the Moore’s law by claiming that the performance per watt of computing grows exponentially at roughly the same rate. Thus, as the number of transistors per die increased, the power density remained relatively constant and in proportion with the area of the die. This enabled the chip manufacturers to increment the clock frequencies without significantly increasing the circuit’s power consumption. Early 21st Century showed signs of Dennard’s Scaling breaking down and the slow-down of Moore’s Law which led computer architects to look for alternate methods of performance growth. This scenario of increasing power consumption and the diminishing performance scaling of uniprocessor architectures led to the advent of new architectures (multi-core CPUs, massively parallel CPUs and special purpose processing devices).

A new era with plethora of parallel processing architectures and programs that take advantage of multiple processors was observed. This growth of multi-core processors researches initially focused on problems with implementing many core processors which lead to the development of Shared-Memory multiprocessor (SMPs) and distributed memory architectures [Figure-1]. The growing prevalence of multicore systems called for a number of parallel applications. Obviously,

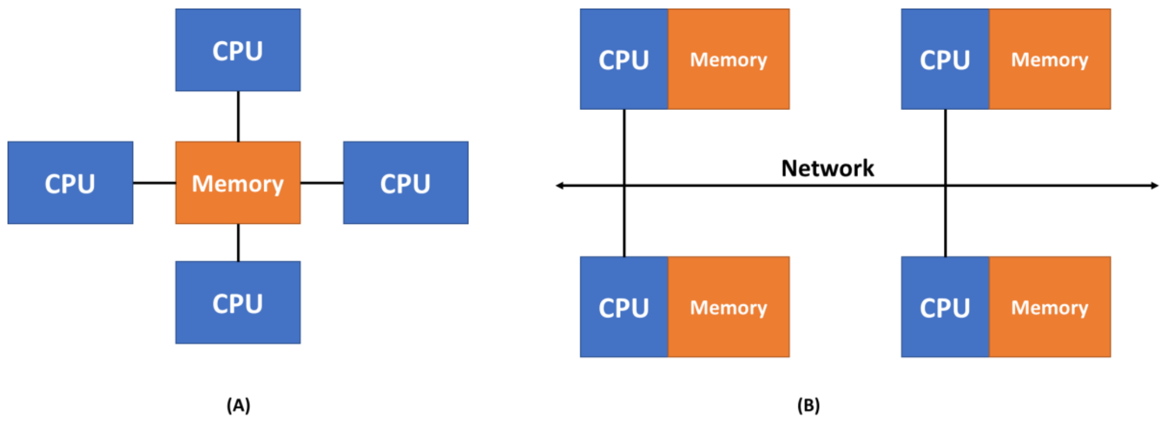


Figure 1: Abstract models of (A) Shared Memory Multiprocessors (B) Distributed Memory Multiprocessors. [18]

there was also hassle in the parallel programming world to come-up with software optimizations that can efficiently utilize the many core processers, but the system as a whole, gave better performance with lesser clock frequency and low heat dissipation.

A globally shared address space makes it easier for the programmers to write parallelizable code for the applications to run on the High-Performance systems. Modern SMPs designs generally contain partitioned global address space (PGAS) where the Most Significant Bits of the address represents which memory controller the address is associated with [1]. It should be noted that in a shared memory model, communication occurs implicitly through load and store of data in the memory allowing the processors to access the most up-to-date data.

Recent years have seen the rise of shared-memory Chip Multiprocessor (CMP) architecture designs. Although, the Chip Microprocessors share a global address space similar to SMP architectures but they also exhibit Non-Uniform Memory Access (NUMA) latencies; SMPs exhibit UMA latencies [Figure-2]. NUMA technology provides a scalable architecture capable of building Massively Parallel (MPP) Systems that provide ease of programmability like the SMPs [2]. In

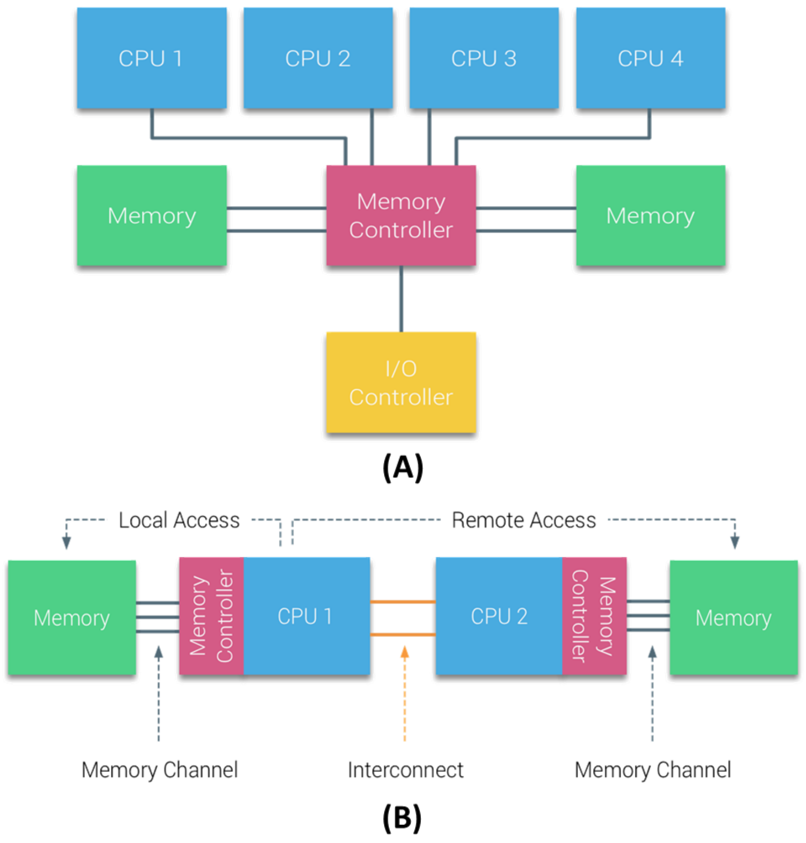


Figure 2: Abstract representation of (A) Uniform Memory Access Architecture (UMA) and (B) Non-Uniform Memory Access Architecture [3]

NUMA multiprocessor design, the memory access time depends on the memory location relative to the processor, i.e. a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors) [4].

Chip Multiprocessors are different from Distributed Memory Multiprocessors in the sense that in CMPs, a single chip consists of all the cores, such that the cores are connected to each other using the on-Chip interconnection network. Figure-3 shows a typical CMP model with 64 nodes,

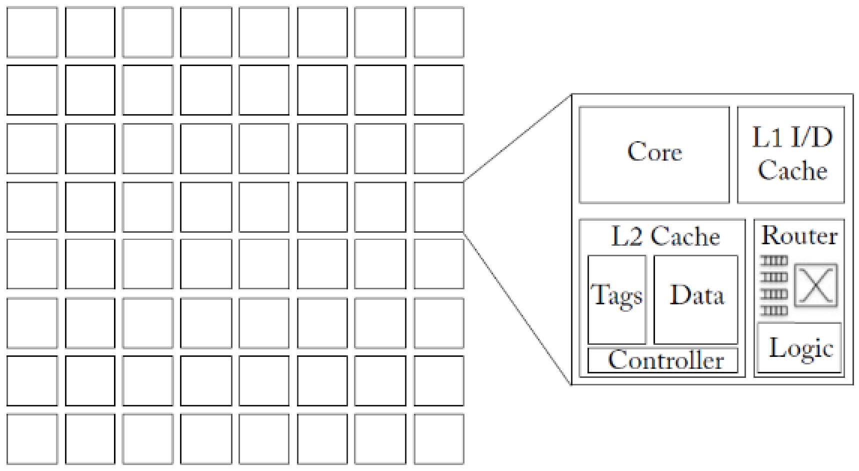


Figure 3: Shared Memory Chip Multiprocessor Architecture. *[1]*

each containing a core, private L1 caches, local l2 cache bank, and other network logic which connects the node to other on-chip nodes [1]. The use of on-chip networks reduces the latency of memory access as it provides much less link-delay as compared to the off-chip connections. Since the cache banks are distributed into multiple nodes, CMPs reflect the need for cache coherency. Cache Coherence Protocols are followed by the coherently working cores to exchange data packets between each other through the on-chip network.

The increase in the number of cores in the network causes a tremendous increase in the network traffic due to rise in the coherence messages, thus making the network performance a bottleneck for the full-system performance [5]. As the modern-day workloads require hundreds of processing units, the research focus has been shifting towards optimizing the performance of the Network-on-Chips. Consequently, there has been tremendous research on on-chip network topologies. Since network topology determines the layout of connections between nodes and channels in the network, it can lead to huge impact on the overall full-system performance.

(Topology determines the number of hops required by the packet to traverse from one node to the other which influences the network latency significantly, thus affecting performance). Practically, since the traversal of data is enabled by the routers and links which incur energy, network topology also shows a direct impact on the power consumption. Also, the way the connections (links) are laid out and the number of these connections also affect the network bandwidth. Thus, the cost of Network Topology is determined by the average degree of nodes and the ease of laying out the NoC on silicon [1]. In the light of the aforementioned topology-cost/system-performance relationship, various Network topology designs have been proposed. Various three-dimensional Topologies have also been proposed, primarily 3D Mesh topologies. 3D ICs perform better as compared to 2D ICs due to reduced interconnection length. Moreover, in a 3D stacked chip processing core layers can be stacked with cache layers which can significantly reduce the memory latency, thus shortening the memory wall [6].

1.2. Motivation

There have been many factors that led to the making of this project. As mentioned in the previous section, with the growing research in the area of Network-on-Chips, articles proposing new methods of NoC layout have been booming. A significant portion has been focused on 3D NoCs lately. Thus, one of the prime reasons for this project was to contribute towards the current research and methodologies by implementing 3D NoC models in a widely used simulation tool so as to serve as a foundation for all future 3D NoC based researches.

Since most of the academic systems research is generally performed using system-simulators, **gem5 simulator** [7] has been chosen as the tool of our choice. We chose gem5 due to its popularity and the **similitude of its full-system simulation results with the actual hardware**, which is generally lacking in other system simulators. Gem5 framework consists of **GARNET** interconnection network model [8] which is a detailed cycle accurate model. Because the state-of-the-art Garnet model essentially consists of only 2D topology model implementations, another motivation was to integrate 3D topologies in the model, thus enabling full-system simulation of workloads on three-dimensional network layout.

Apart from the above-mentioned factors, the constant support and guidance from my graduate advisor worked as factor in itself to help stick to the project goals and objectives regardless of the hardships involved.

1.3. Report Organization

In this project report, a detailed information about the implementation of 3D mesh and 3D stacked-Mesh NoC topologies on gem5 simulator. In the adjoining section, i.e. Related Work, the state-of-the-art design choices for network topologies, their observations and respective limitations are discussed. The following section provides a small background on GARNET network model. Section-4 provides a detailed description of the project implementation including the discussion on our approach of optimizing the number and position of vertical links in the network. Section-5 talks about the simulation results. Finally, Section-6 provides concluding remarks on the project with a light over potential future work opportunity.

2. RELATED WORK

Computer Architecture research has always been pre-dominantly focused on powering computer systems. With the growing needs for smaller, more efficient and powerful systems, there has been tremendous research done in the area of Systems-on-Chip (SoCs). SoCs consists of different system components like processor cores, l1 caches, l2 caches, network interface, etc. all on a single chip [9]. Figure-4 shows an abstract representation of a Multiprocessor System-on-Chip.

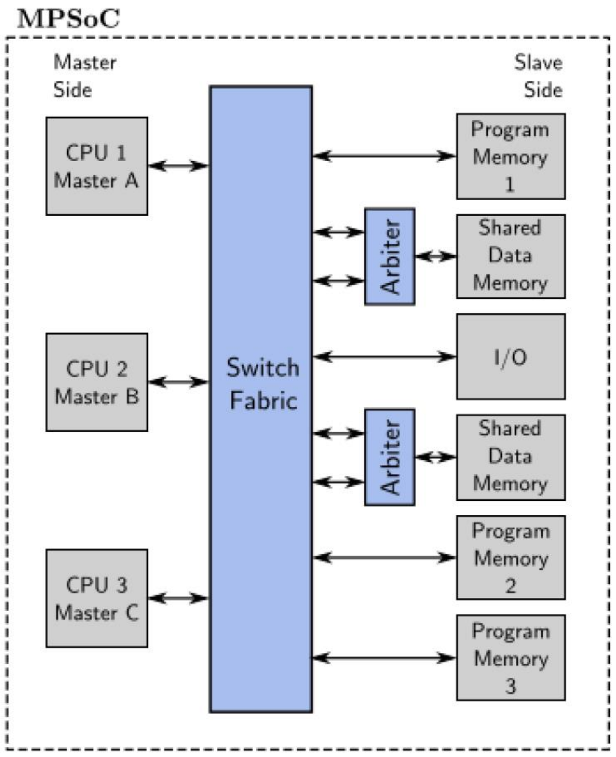


Figure 4: Multi-Processor System-on-Chip representation. It incorporates cores, caches, interconnect (switch fabric) and the main memory. *[5]*

2.1. 3D Die-Stacking Techniques

With the research in 3D ICs, many multi-layer SoC designs are possible. Loh et al. describe a few 3D die-stacking designs [10]. The authors inform about the design choices concerning the size of vertical links (also known as Through-Silicon Vias) in a 3D die. According to the authors, a very fine d2d via enables very fine partitioning of processor structures between multiple layers whereas larger via reduces the available inter-die bandwidth for each unit area. They mention that the TSV size ranges from (10 μm x 10 μm) to (1 μm x 1 μm); since they are significantly larger than an individual transistor, a finer TSV link is preferred for many 3D organizations. As for this project, because the implemented model works on an abstracted view and concerns only with the provision of the functionality, TSV size is not taken into consideration. Although, it would be a fair assumption to consider fine-pitched TSV links in our model.

Loh et al. also discuss about various 3D microarchitecture designs depending on the granularity of the unit to be stacked and the re-design effort from the 2D designs. Figure-5(A) represents the stacking of macroscopic blocks. For example, stacking of cache blocks on the top of CPU cores or stacking separate cores in a multicore design. The advantage of this approach is the reduction in a multicore processor's core-to-core interaction paths. Although, this approach significantly uses the existing 2D design efforts, thus reducing fewer critical wires in the SoC and hence limiting the performance and power benefits. Figure-5(B) represents a design choice where the functional units are stacked on the top of each other as a 3D approach, e.g. ALU-on-ALU (for faster bypass), DL1-on-ALU (for faster loads), etc. This approach requires a medium level effort in re-designing. It would involve re-creating floorplans and timing paths with the re-use of 2D FUBs. In terms of potential benefits, this approach leads to performance improvements and power reductions simultaneously. The third approach [Figure-5(B)] for 3D stacking is one abstraction level down from the second approach. In this approach, individual Functional Unit blocks are split across multiple layers for a potential gain in performance. For example, stacking-up the entries of

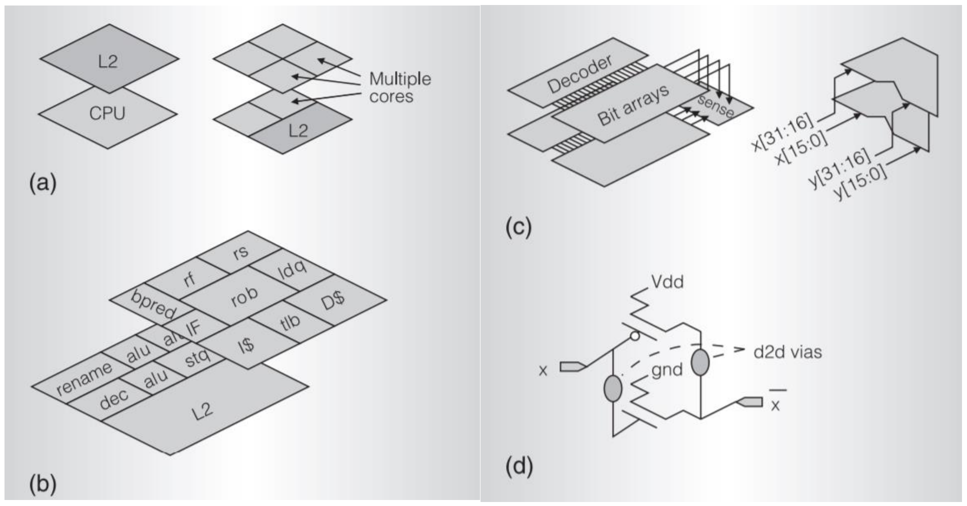


Figure 5: 3D IC designs based on granularity. (A) Entire core stacked (B) Stacked Functional unit blocks (C) Stacked logic gates and (D) Staked transistors. *[10]*

processors' reservation stations, bit-slicing of functional units, splitting caches' work-lines, etc. The potential benefits of this approach include: elimination of intra-block wiring, leading to significant power and performance benefits for wire-dominated blocks. Obviously, huge re-designing efforts are needed such as, 3D circuit designs and layout tool development, etc. According to the authors, the finest possible 3D design can be done at transistor level [Figure-5(D)]. This approach has the potential of providing large reductions in areas, latency and power consumptions, thus providing maximum performance benefits. The problem with this approach is the strenuous design efforts. Since at this level of abstraction, it is very difficult to reuse existing 2D resources, this approach is very time consuming and costly in terms of labour. Because the focus of the project would be on the Architectural aspects of the SoCs, the project exploits the first approach, i.e. the stacking of the macroscopic blocks (cores and cache banks) on top of each other.

Also, gem5 full-system simulator platform, (for which we are implementing the 3D stacked approach) works on the component level, exploring the first approach suits our interests.

2.2. 3D Architectures for Multi-core Processors

Loh et al. [10] also briefly discuss 3D architectures for multi-core processors. According to the authors, as the system starts to scale, there is always an incremental effect of wire-delays in terms of longer memory access times or non-uniform memory access times for a very large CMP. The wire-delay issue can be reduced by using NUCA cache architectures [11] in CMPs. Loh et al. claim that the use of d2d low latency vias in 3D NoCs help to increase the memory locality even further and hence reduces the cache access time. They conducted experiments to claim that the 3D version provides 50% reduction in the cache access time than the 2D NUCA architecture. The reduction in the cache latency leads to reduction in network contention, thus lowering the power consumption. Regarding this, Gardea et al. [12] also informed that 3D NoCs helps in reducing the network diameter leading to a decrease in the packet latency, this is due to the result of increased connectivity of the routers in 3D NoC compared to the 2D NoCs. Feero and Pande [13] laid out differences between a 3D NoC and a 3D stacked NoC [Figure-6]. According to the authors, a 3D mesh is an extension of its 2D predecessor, such that multiple layers are connected using the vertical same as the individual links on each layer. In a 3D stacked mesh, instead of normal vertical links, a small low latency bus runs through the entire vertical distance of the chip.

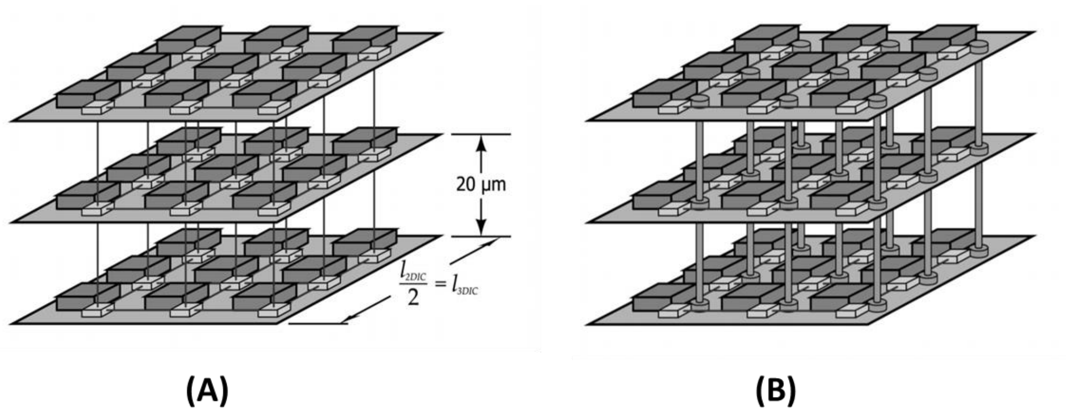


Figure 6: Mesh-based 3D NoC Architectures. (A) Three-dimensional mesh (B) Stacked mesh. *[13]*

In this project, both the mesh-based 3D NoC types [in Figure-6] have been implemented. In our version of stacked-mesh, the first layer is a core layer, consisting of only processing cores, the rest of the layers consists of l2-cache nodes. Although the gem5 simulator does not support implementation of bus-interconnect in GARNET, the framework enables the user to set the latencies of each link in the network. Thus, for effectively the same effect as the bus in 3D stacked architecture, the vertical TSV-links have been used with lower latencies than rest of the NoC links.

Although 3D ICs generally out-perform two dimensional ones, one of the biggest concerns regarding the packaging of layers is in terms of thermal aspects. According to Xu et al. [14], since processors draw majority of power from the chip, stacking of multiple processors over one-another lead to huge amount of heat dissipation. They claim that 3D NoC designs can show typically over 29°C temperature raise than a 2D NoC design. Based on the above stated facts, the authors suggest that nearly half of the area of the die should be allotted to the cores and the rest of the area should be consumed by the network components, shared caches and memory controllers and thus the 3D IC is partitioned into layers such that particular type of components are allotted to their respective layers only (e.g. cores are placed in only one layer, generally the top layer due to its connection with the heat sink). The 3D-stacked Mesh model in our project was implemented keeping in view the point-of-view of the authors.

2.3. Through-Silicon Vias Placement Criteria in 3D NoC

Xu et al. also inform that as we scale the size of the 3D NoC it becomes difficult to connect all the corresponding nodes to the inter-layer TSV links due to high manufacturing costs and limited chip area available for interconnect. Also, since the maximum performance would be gained by full "layer-layer" connections only, this becomes an optimization problem of choosing the minimum number of TSV nodes that can provide optimal NoC performance. The authors also discuss their method of selecting placement positions for the TSV links. They've used "Average Hop Count" as their metric for NoC performance evaluation. They provide some stats regarding the combinations of TSV placement: for an 8x8 mesh, considering half the TSV connections lead to 1.83e+18 possible optimal-hop count configurations, quarter TSV connections lead to around 488,526,937,079,580 possible optimal-hop count configurations. The authors in [15], further explain their methodology of exploring the design space for optimal TSV placement. They compare 16 and 8 number of TSV connections only, for an 8x8 mesh design, because of relatively very smaller number of possible configurations (between 1000 and 2000). Their methodology is to explore the entire design space to choose the most optimal TSV placements for the NoC. Of course, because the possible configurations are still very large to explore, the authors divide the NoC into smaller sub-sections in order to reduce the design space but still keep a decent NoC configuration. For our 3D stacked mesh model, the optimal TSV placement problem is handled using a simpler approach. Our approach places N vertical TSV links for every NxN mesh design based on the average hop-count metric similar to the authors. But, instead of comparing all the possible configurations, our algorithm uses a branch-and-bound approach to choose indices of the TSV links according to a set of rules formulated for another placement problem. Thus, enabling a faster simulation for 3D stacked Mesh NoCs in gem5.

3. GARNET NETWORK MODEL

GEM5 simulator framework consists of a very accurate GARNET: Network-on-chip simulator model useful for simulating varied interconnect systems. This section provides a brief background on the model and discusses the parameters needed for our implementation of 3D NoC.

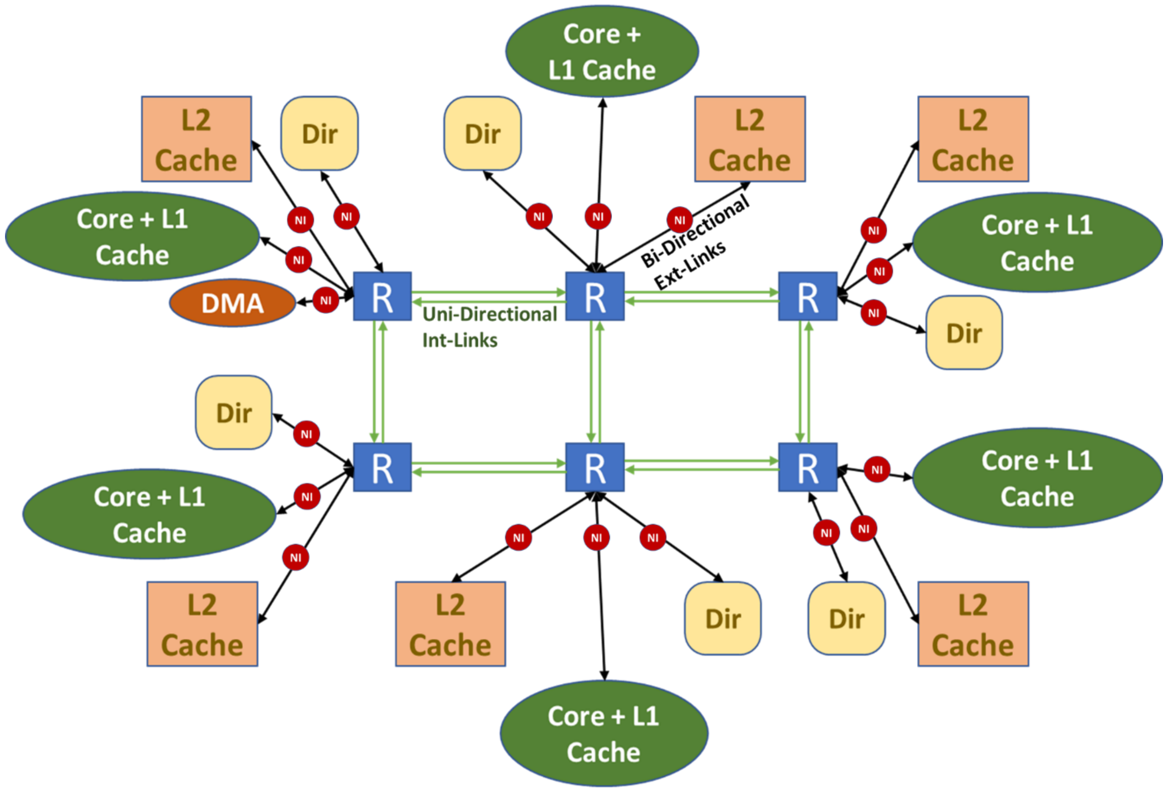


Figure 7: An Abstract View of GARNET Topology Model. *[16]*

Garnet contains a Topology class in gem5's ruby memory system model that enables the user to design and implement their heterogeneous topology designs [17]. As shown in the Figure-7, the topology class enables each router object to connect to the instances of other network components (cores, l1/l2 cache banks, directory nodes, DMA controllers, or more router objects). These components are connected using the Link instances of the classes defined under Garnet Link infrastructure. The router-to-router connections are laid out using “Internal links” which are instances of “GarnetIntLink” class. All internal links are uni-directional. Connections to other components are laid out using the “External links” which are instances of “GarnetExtLink” class. The external links are bi-directional. Each link instance contains two parameters: “src_output” and “dst_output” where the user can specify the output and input ports for the source and destination router instances. The structure of the Interconnection Network can be designed via python files where the user can specify how the components are connected. Once the layout is established by the users, the “GarnetNetwork” class generates a network object which instantiates objects of other network components (network interface, routers, links). The “Topology.cc” file calls the methods to attach external and internal links for Network Interface-routers and routers-routers respectively according to user’s NoC layout. Each Network Interface (NI) object, an instance of “NetworkInterface” class is connected to one coherence controller on one side and to a router on the other. Of course, the GARNET framework also models the Router Microarchitecture, flow control and routing infrastructures but they have not been discussed due to their limited application for the user in this project.

Garnet also enables the user to set parameters independently for each router and link in the network. For example, the *latency* parameter in Router sets the latency for each router object. Some of the useful parameters for Link objects are:

1) *latency*: Sets the latency of packet traversal through the link.
2) *weight*: This parameter is useful for routing purposes. Each link has a weight associated with it which is used by the routing table to decides routes that the packet will follow. Note that this parameter does not affect the latency of the link.
3) *bandwidth_factor*: This works as a bandwidth multiplier. Default value = 1

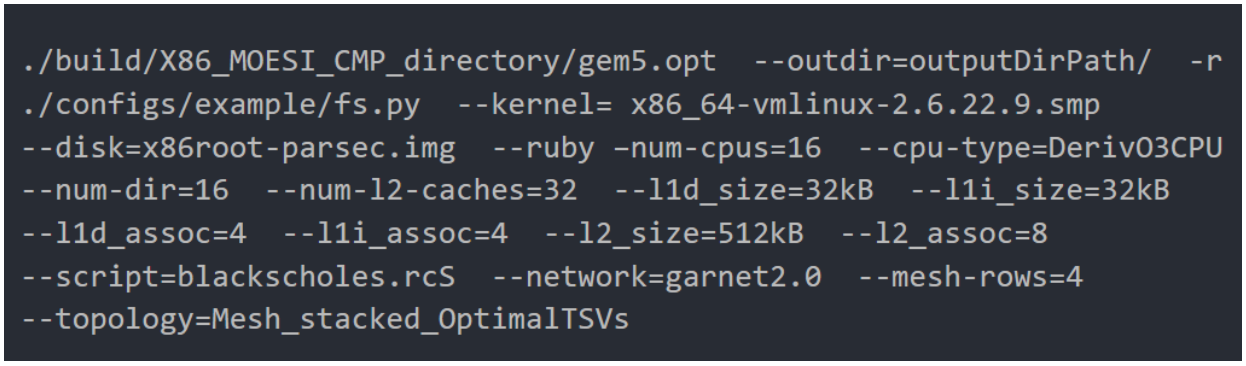
Link bandwidth = bandwidth multiplier * endpoint_bandwidth

Unfortunately, this parameter is not used by the Garnet Network model but is only used by Simple network model in gem5. Garnet does not provide the feature to set individual link bandwidth.

As mentioned before, one of the prime reasons for choosing gem5 as the simulation tool of choice was its ability to integrate network models in full-system integration. Figure-8 shows the full-system simulation command with its corresponding parameters, that we use to simulate our 4x4x3 3D stacked model with optimal TSVs. (Notice that the command line also passes some network parameters like --*num-cpus*, --*num-l2caches*, --*num-dirs*.)

Also, our NoC code can run on both the network models viz. Garnet (--*network=garnet2.0*) and Simple-Network (--*network=simple*).

Command line:



```
./build/X86_MOESI_CMP_directory/gem5.opt  --outdir=outputDirPath/  -r
./configs/example/fs.py  --kernel= x86_64-vmlinux-2.6.22.9.smp
--disk=x86root-parsec.img  --ruby -num-cpus=16  --cpu-type=DerivO3CPU
--num-dir=16  --num-l2-caches=32  --l1d_size=32kB  --l1i_size=32kB
--l1d_assoc=4  --l1i_assoc=4  --l2_size=512kB  --l2_assoc=8
--script=blackscholes.rcS  --network=garnet2.0  --mesh-rows=4
--topology=Mesh_stacked_OptimalTSVs
```

Figure 8: Gem5 Command for full-system simulation.

4. PROJECT IMPLEMENTATION

This section discusses our approach to implement the aforementioned 3D Network-on-Chip designs on the gem5 GARNET Network model. We also demonstrate our branch-and-bound approach for calculating the optimal number and position of TSV links for the stacked mesh architecture.

The “configs/network/Network.py” file generates an object of Network class. One of the parameters that are passed-on to this object is an object of one of the *Topology-defining* classes implemented in this project (“Mesh_3D”, “Mesh_stacked”, “Mesh_stacked_withOptimalTSVs”). These *python* classes for defining the structure of the three-dimensional NoCs were implemented based on the study of pre-implemented 2D Mesh designs in the simulator. The pseudocode in listing-1 demonstrates how the network components are initialized and NoC design is established for simulation.

All external controllers generated by the Memory protocol are passed down as a list to the class object which are used by the Topology class as network nodes. The command-line arguments to the network are used to declare and initialize some important parameters to the network, for example, network dimensions, number of routers, link-latencies, router latencies, etc. To check for the viability of the user-provided parameters according to the topology structure, various sanity checks have been put into the code so that the execution could be stopped before a buggy simulation could start. For example, the number of nodes in the network should be a positive multiple of the total number of routers in the Network (since all the nodes are connected to each other using routers, the number of routers should at least be equal to the total nodes in the network). As informed in the previous section, each router is an instance of Router class defined in GARNET

Listing-1: <u>Pseudo-code for defining 3D NoC layout</u>

```
function setTopology():
    Initialize network nodes list   # contains L1 controllers, L2 controllers,
                                    # Directory controllers, DMA controllers
                                    # generated by gem5.
    Set number of mesh-rows
    Initialize number of routers    # for Mesh-3D: num-routers = num-cpus
                                    # for Mesh-stacked: num-routers = l1 + l2 ctrls
    Initialize num-cols
    Initialize num-layers

    Define general link and router latencies

    sanity check for user input-parameters

    For num-routers, declare instances of Garnet Router class

    Setup external links    # each router is connected with external controllers
                            # (cache controllers, directory nodes, DMA controllers)

    # Setup layer-vise Internal links
    for each layer:
        # Connect routers per layer

        # Setup Intra-column links (West-East and vice versa)
        for each row:
            for each col:
                connect routers   # weight = 1

        # Setup Intra-row links (North-South and vice versa)
        for each col:
            for each row:
                connect routers   # weight = 2

    Generate TSV link positions

    # Setup Intra-layer links
    for (row, col) TSV index positions:
        for each layer:
            connect routers     # weight = 3

    Send router and external/internal links to gem5
```

Network model. Therefore, after passing the checks, num-routers number of router objects are generated, followed by connecting them to external controllers using the ext-link objects of External Link class. These external links are then stored in the network object’s external links list. Following this, internal-connections are made for every router in the network in order to define the desired Network-on-Chip layout. For keeping the implementation simple, all the horizontal connections (intra-link) are paved before laying-out the vertical (inter-link) connections. Notice in the code listing-1 that each block of router-router connection part of the program defined per-link weights. Weights are used by the network to determine how the Weight-based routing is intended. For the simplicity purposes, we’ve focused on X-Y-Z routing, but Garnet provides a very simplified way for implementing and inducing the user-defined routing during the network simulations. After all the internal unidirectional connections are laid-off, these link objects with their corresponding connection information are also dumped in the network object’s internal links list.

The following sub-sections briefly informs about the variations between our implemented 3D-NoC models in terms of node arrangement and type/number of inter-layer links.

4.1. 3D Mesh NoC

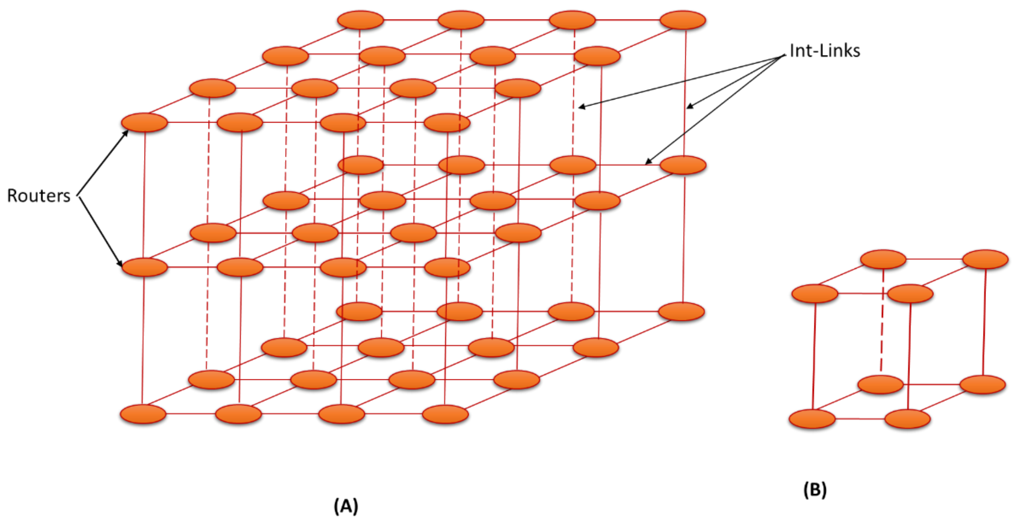


Figure 9: 3D-Mesh NoC Designs. (A) 4x4x3 3D-Mesh (B) 2x2x2 3D-Mesh

Our 3D Mesh implementation has a structure similar to the two-dimensional Mesh designs, just extended to the third dimension. For this model, the number of routers in the network equals the number of processing cores passed as an argument by the user. Thus, an NxNxM 3D mesh consists of N*N*M processing elements and the external controllers are distributed equally among these nodes. All the routers present in the network are of equal latency. Every link (external or internal) is of equal bandwidth and latency.

Figure-9 shows the structural representation of the 3D Mesh. For the sake of visual simplicity, the external controller nodes (which are connected to each router via external links) have not been shown. Also, in actuality, for every int-link shown in the figure, two unidirectional router-router links are present in the network. Although the default structure of our 3D Mesh is symmetrical, i.e. number of rows in every layer is equal to number of columns, the code is written such that it is fairly easy to form asymmetric 3D Mesh designs with any number of layers as desired by the user.

4.2. 3D-stacked Mesh NoC

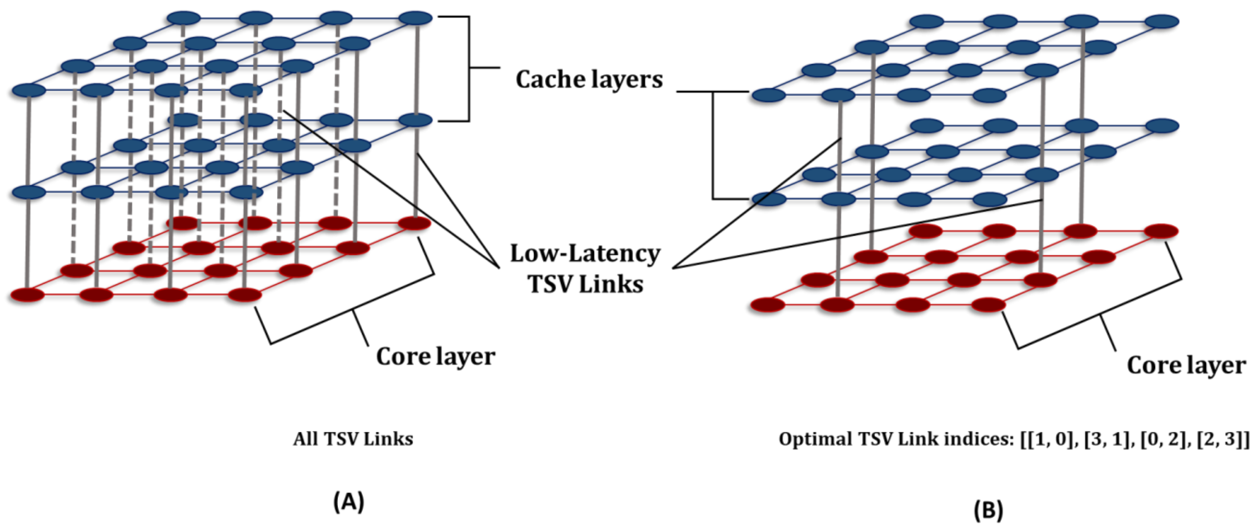


Figure 10: An Abstract Layout of a 4x4x3 3D-stacked Mesh Network-on-Chip Interconnect. (A) All low-latency TSV Links are connected. (B) Optimal Number of low-latency TSV Links (four, in this case) are connected.

The 3D-stacked Mesh layout [Figure-10] implementation has a few subtle differences as compared to the 3D Mesh design. In the latter, all the computing cores are connected to each other in a 3D fashion and the other Network node-types, for example, cache-banks and directory nodes are connected directly to the individual cores. Unlike this, the L2 cache-nodes in the former have been separated from the processing core-nodes and are placed in separate layers. Thus, alike conventional stacked-mesh architectures, our model also consists of one core-layer and two cache-bank layers connected to each other via low-latency TSV links. Note that we have restricted the number of layers to only three because greater than that is generally not practical due to temperature issues and die-area limitations. Although, the code is written such that it is fairly easy to make changes to the number of layers.

Following many state-of-the-art researches regarding the search for optimal number of TSV links, we have introduced an implementation for the same in our model also [refer Figure-10 (B)]. As shown in code listing-1, the Topology file calls another sub-routine that returns a list of ‘N’ TSV indices for an ‘NxN’ 3D mesh-layer such that the average hop-count from a node to the nearest TSV link is optimal. The following sub-section describes the approach for the same.

4.3. Determining Optimal TSV-link indices

The problem of finding the optimal TSV link is a space exploration optimization problem of getting the least number of TSVs for the maximum possible performance. Typically, that involves finding the index positions in the grid such that the average packet hop count is minimized. We observed that there exist configurations containing only ‘N’ number of TSV links that fulfil the above criteria. Figure-11 shows one such configuration for a 4x4 Mesh and an 8x8 Mesh. It was also noted that although the placement of TSV links might differ, all the configurations that represent TSVs placed in a sparse manner on the grid (such that no two TSVs are linked in the same row, column or diagonal), provide the same minimum average hop count. Thus, instead of exploring all the possible configurations, our implementation uses a branch-and-bound approach

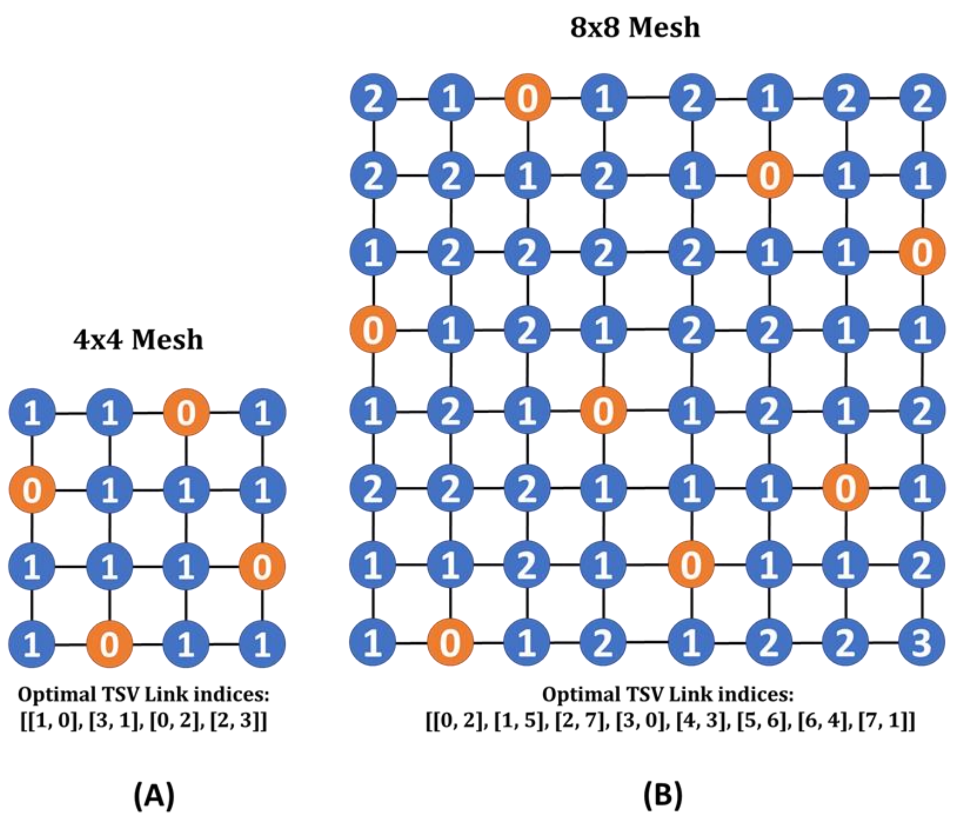


Figure 11: Packet Hop values from each Network-Node to a nearest TSV link in (A) 4x4 Mesh and (B) 8x8 Mesh.

to generate only one such configuration that is in accordance with the above stated placement criteria. The approach for the required algorithm is inspired from N-Queens Problem, a classic coding question, as it was very similar to the TSV placement problem. Akin to the N-Queen problem, we place the TSV links on the mesh under the constraint that only a single TSV link should be present in the entire row, column and diagonally. This approach restricts the number of TSVs in the grid and automatically places them in the position of minimum possible hop count for a node. As can be seen from the Figure-11 (A), if the TSV links are connected to nodes at indices: (1, 0), (3, 1), (0, 2) and (2, 3), all the other nodes in the network are only 1 hop count away from the nearest TSV. The hop-count for nodes in 8x8 mesh [Figure-11(B)] is also minimal considering the size of the network.

Of course, there exists other approaches for finding the TSV placement indices and the number of TSVs to be placed might also differ. The project code is written such that a sub-routine is called to determine the optimal TSVs, therefore, any TSV optimization approach can be used in our stacked-Mesh model, as long as the subroutine returns the lists of indices where the vertical TSV links need to be connected.

5. SIMULATION RESULTS

Full-system simulations were carried out on PARSEC Benchmark Suite to evaluate the validity of our 3D NoC models. We collected system and network stats to compare two NoC designs to verify whether the simulations on the benchmarks produce the expected results. In this section, we discuss a few of those simulation results.

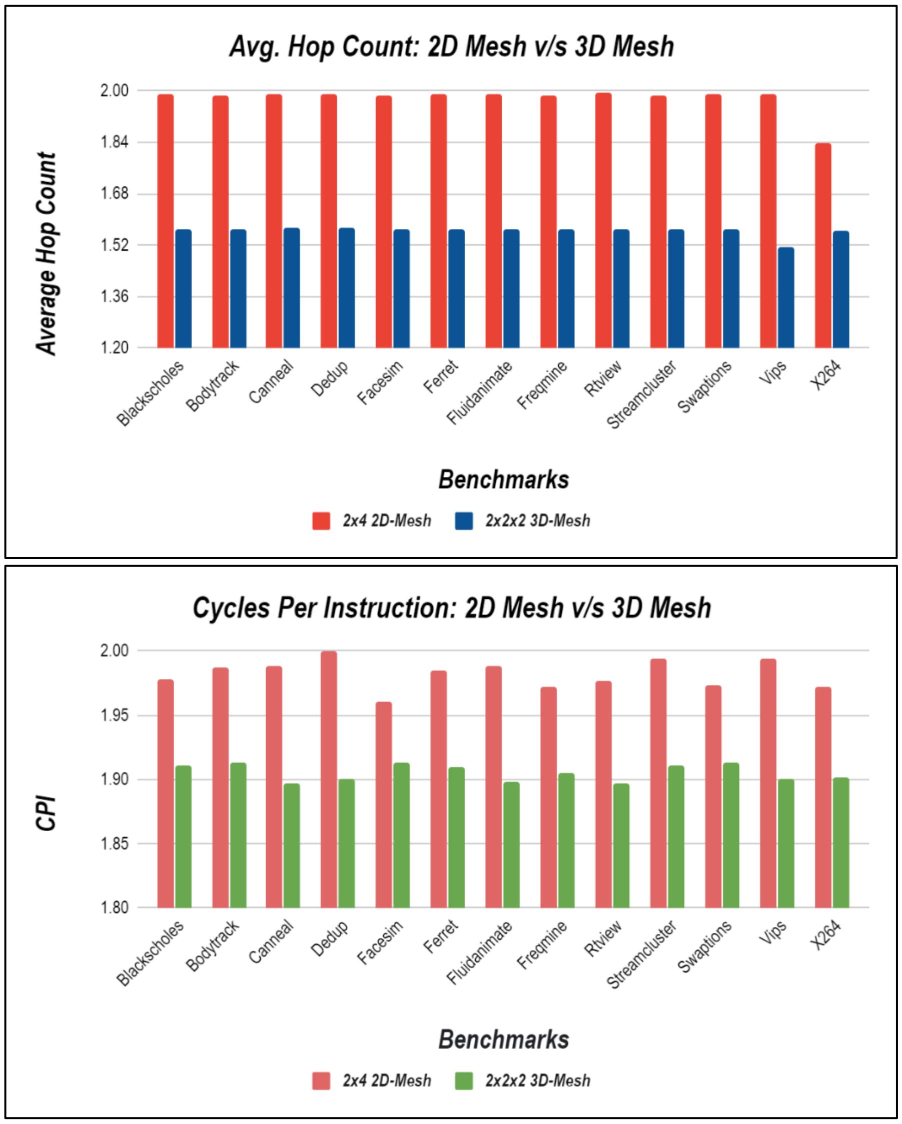


Figure 12: Distributions Comparing Performance of 2D and 3D Mesh results. (Top) Represents Average Hop Count values for 2x4 2D-Mesh and 2x2x2 3D-Mesh. (Bottom) Represents values for Average CPI for 2x4 2D-Mesh and 2x2x2 3D-Mesh.

Figure-12 compares the simulation results between 2x4 Mesh and 2x2x2 3D-mesh NoC designs. It can be seen that 2x2x2 3D-mesh design leads to about 21% reduction in the average hop-count as compared to the 2x4 2D Mesh design [Figure-12 (top)]. 3D network also leads to better performance than the 2D mesh as can be verified by the average CPI for the workloads [Figure-12 (bottom)]. As both the NoC design consists of equal number of computing-units, better performance of the 3D model conforms with the expected behaviour and aides in verifying our implementation of 3D NoCs.

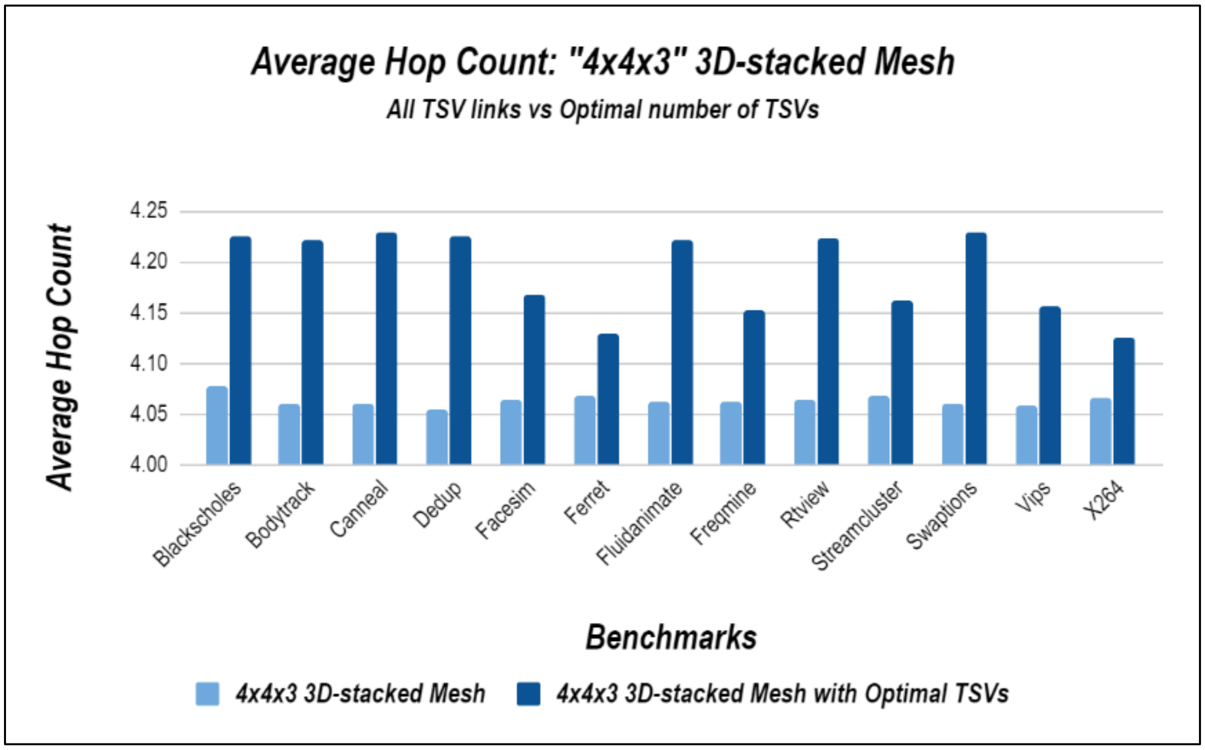


Figure 13: Distribution representing the Average Hop Count comparison between 4x4x3 3D-stacked Mesh models w/ all TSV Links connected and w/ Optimal number of TSVs connected.

Figure-13 aides in verifying the implementation of our branch-and-bound algorithm that calculates the optimal TSV indices. From the distribution it can be clearly seen that although the number of TSV links was considerably less (only a quarter of the total number of TSVs possible), the optimized version of the NoC design only showed an increase of around 3-4% in the average hop-count values compared to the naïve 3D-stacked Mesh. These results also follow the anticipated results that the optimized approach should be worse in computing performance. Also, a very slight increment in the hop count reflects the efficiency of our approach.

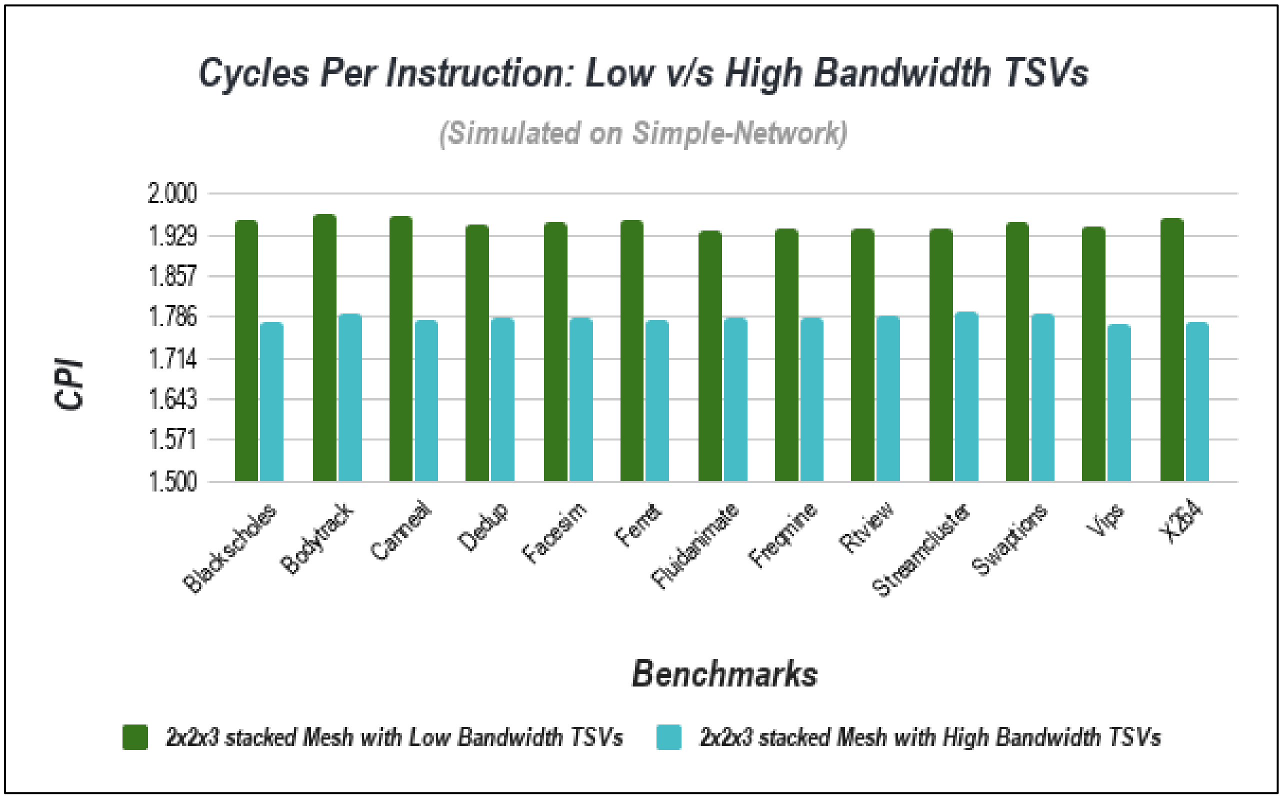


Figure 14: Performance Comparison between 2x2x3 stacked Mesh with low-bandwidth TSV and another 2x2x3 stacked Mesh with high bandwidth TSVs.

A few 3D stacked Mesh related researches represent TSV vertical links as low-latency and high-bandwidth internal links. Since the GARNET model does not include the feature for setting individual link bandwidths, the 3D stacked model was simulated on gem5's "SimpleNetwork" model. For the simulation, the bandwidth of the TSV links was set thrice of the bandwidth of rest of the internal links in the Network. Figure-14 shows the average CPI results comparing the high-bandwidth TSV 2x2x3 model with the low-bandwidth version. It can be seen that the high bandwidth NoC gives around 8-9% smaller CPI than the low-bandwidth version, thus reflecting that the high-bandwidth version has a higher performance, which conforms with the expectations. Moreover, the routing algorithm used for simulation is X-Y-Z Routing, it means that the packets would have travelled through the low-latency links before accessing the required cache-banks through the TSVs. It is fair to assume that a different routing algorithm that can explore the high-bandwidth property of TSVs is bound to provide much better performance than the normal 3D NoC design.

To further validate the results of the simulations, we compare our results with the metrics provided by Ferro and Pande [13]. The authors provide a metric to confirm the total number of links in the mesh-based NoC layout.

$$\boldsymbol{links = N_1N_2(N_3 - 1) + N_1N_3(N_2 - 1) + N_2N_3(N_1 - 1)}$$

where N_i represents the number of routers in ith dimension. Since the total number of internal links can also be correctly classified by the above formula, this can be considered as a proof of correct connectivity of our NoC layout implementation. The authors also provided a metric to calculate the average number of hop-count in the mesh-based networks.

$$\boldsymbol{hops_{NoC} = \frac{n1n2n3(n1 + n2 + n3) - n3(n1 + n2) - n1n2}{3(n1n2n3 - 1)}}$$

where n_i represents the number of nodes in ith dimension. Based on the above formula, it was observed that the simulation results on our 2x2x2 3D Mesh NoC provided 5-8% smaller values of average hop-count, which is a very minimal difference considering the above metric was provided for no-load latency networks. It should be noted that although our 3D-stacked models would follow the formula for calculation of total links, it would not be in accordance with the above metric for average hop-count calculation, this is because, the "nodes" in the formula refer to the processing-cores only and our 3D-stack implementation consist of only one core-layer.

The results provided in this section demonstrate the fact that the implemented 3D NoC models were simulated successfully on Garnet framework in the gem5 simulator, and since the results agree with the accepted notion of 3D NoC performance, it verifies the validity of the work performed.

6. CONCLUSION & FUTURE WORK

In this project report, we demonstrated the need for Network-on-Chips and informed about the recent researches in the field of 3D NoC designs. The report provided description of our implementation of three-dimensional Mesh designs for Network-on-Chips to be simulated on GARNET Network model in the gem5 simulator. We implemented three variations of 3D-mesh NoC designs: *3D mesh*, *3D-stacked Mesh* and *3D-stacked Mesh with Optimal TSV links* and discussed the subtle differences in their structure. All the implemented network models were simulated on the Garnet platform and the results were discussed and analysed which stand as a proof for the successful execution of the model and validates the correctness of the design.

More decisive methods are needed to validate the implementation of our 3D NoC models. Although there exists many accurate 3D NoC simulators, e.g. Booksim simulator, but since these simulators are network-only and do not provide full-system simulation facility, they do not stand as good candidates for our model validation. Moreover, most of the other simulators do not share the same level of implementation details as the gem5 simulator. Also, differences in cache coherence protocols and programming workloads add to the complexity of validating our results against pre-existing 3D NoC implementations. Simulations on many 3D NoC sizes along with their result analysis has been left for the future due to the lack of time (i.e. the simulations are more time consuming for large NoC structures and also limited to the computing prowess of the server). In future work, integration of our NoC models with power models (e.g. MCPAT) might prove important as the users would then be able to estimate the feasibility of their design in terms of power consumption, die-area consumptions, etc. also along with the computing performance that our model outputs. As stated before, many more sophisticated approaches exist for the optimal TSV placement problem, their future-application on our 3D stacked NoC model would enable the researchers to compare their impact on the network performance, which is otherwise a cumbersome task. Of course, there exists many different 3D NoC designs other than Mesh designs which might prove better for specific workloads, their future-implementation would help in expanding the scope of the project.

In conclusion, our work enables the provision of 3D NoC-based full-system simulation feature on the gem5 simulator and our results validate its correctness.

REFERENCES

[1] N. E. Jerger, T. Krishna and L.-S. Peh, "On-Chip Networks," in *On-Chip Networks*, Morgan & Claypool, 2009.

[2] X. Guo and H. Han, "A good data allocation strategy on non-uniform memory access architecture," *IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pp. 527-530, 2017.

[3] F. Denneman, "frankdenneman.nl," 7 July 2016. [Online]. Available: https://frankdenneman.nl/2016/07/07/numa-deep-dive-part-1-uma-numa/. [Accessed Friday May 2020].

[4] "Non-uniform memory access," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Non-uniform_memory_access#cite_note-nyu-numa-1. [Accessed Friday May 2020].

[5] H. Suresh, "Optimization of Communication Traffic in Hammer Protocol Using 3D Electronic Mesh Network On Chip," 2016.

[6] C. C. Liu, I. Ganusov, M. Burtscher and S. Tiwari, "Bridging the Processor-Memory Performance Gap with 3D IC Technology.," *IEEE Design and Test of Computers,* vol. 22, no. 6, pp. 556-564, 2005.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit.*, vol. 39, no. 0163-5964, pp. 1-7, 2011.

[8] N. Agarwal, T. Krishna, L.-S. Peh and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," *IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA*, pp. 33-42, 2009.

[9] anysilicon, "anysilicon," 10 July 2019. [Online]. Available: https://anysilicon.com/what-is-a-system-on-chip-soc/. [Accessed Saturday May 2020].

[10] G. H. Loh, Y. Xie and B. Black, "Processor Design in 3D Die-Stacking Technologies," *IEEE Micro*, vol. 27, no. 3, pp. 31-48, May-June 2007.

[11] C. Kim, D. Burger and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches.," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, California, Association for Computing Machinery, 2002, pp. 211-222.

[12] J. Gardea, Y. Jin, A.-H. Badawy and J. Cook, "Performance Evaluation of Mesh-based 3D NoCs.," in *Proceedings of the 10th International Workshop on Network on Chip Architectures (NoCArc'17)*, New York, NY, USA, Association of Computing Machinery, 2017, pp. 1-6.

[13] B. S. Feero and P. P. Pande, "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 32-45, Jan 2009.

[14] T. C. Xu, P. Liljeberg and H. Tenhunen, "A study of Through Silicon Via impact to 3D Network-on-Chip design," in *2010 International Conference on Electronics and Information Engineering*, Kyoto, 2010.

[15] T. C. Xu, P. Liljeberg and H. Tenhunen, "Optimal number and placement of Through Silicon Vias in 3D Network-on-Chip," in *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, Cottbus, 2011.

[16] T. Krishna, "A Detailed On-Chip Network Model Inside a Full-System Simulator," Monday, September 2017. [Online]. Available: https://pdfs.semanticscholar.org/c1e9/0beac857ce1af1a531b6538804e71efdef05.pdf. [Accessed Tuesday May 2020].

[17] J. Lowe-Power, "gem5 Documentation," gem5, [Online]. Available: http://www.gem5.org/documentation/. [Accessed Tuesday May 2020].

[18] N. Ploskas and N. Samaras, GPU Programming in MATLAB, Elsevier Science, 2016.