ADVANCED MANAGEMENT TECHNIQUES FOR MANY-CORE COMMUNICATION SYSTEMS

SHARIFA AL KHANJARI

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

SCHOOL OF COMPUTING SCIENCE

COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

SEPTEMBER 2016

Abstract

The way computer processors are built is changing. Nowadays, computer processor performance is increased by adding more processing cores on a single chip instead of making processors larger and faster. The traditional approach is no longer viable, due to limits in transistor scaling. Both industry and academia agree that scaling the number of processing cores to hundreds or thousands on a single chip is the only way to scale computer processor performance from now on. Consequently, the performance of these future many-core systems with thousands of cores will heavily depend on the Network-on-Chip (NoC) architecture to provide scalable communication. Therefore, as the number of cores increases the locality will only become more important. Communication locality is essential to reduce latency and increase performance. Many-core systems should be designed such that cores communicate mainly to the neighbouring cores, in order to minimise the communication cost.

We investigate the network performance of different topologies using the ITRS physical data for the year 2023. For this reason, we propose abstract synthetic traffic generation models to explore the locality behaviour in many-core NoC systems. Using the synthetic traffic models —group clustering model and ring clustering model —traffic distance metrics may be adjusted with locality parameters. We choose two many-core NoC architectures —distributed memory architecture and shared memory architecture —to examine whether enforcing locality on different architectures may have a diverse effect on the network performance of different topologies.

Distributed memory architecture uses the message passing method of communication to communicate between cores. Our results show that the degree of locality and the clustering model strongly affect the performance of the network. Scale-invariant topologies, such as the fat quadtree, perform worse than flat ones because the reduced hop count is outweighed by the longer wire delays.

In shared memory architecture, threads communicate with each other by storing data in shared cache lines. We design a hierarchical cache model that benefits from communication locality because many-core cache hierarchy that fails to exploit locality may end up having more cores delayed, thereby decreasing the network performance. Our results show that the locality model of thread placement and the distance of placing them significantly affect the NoC performance. Furthermore, they show that scale-invariant topologies perform better than flat topologies. Then, we demonstrate that implementing directory-based cache coherency has only a small overhead on the cache size. Using cache coherency protocol in our proposed hierarchical cache model, we show that network performance decreases only slightly. Hence, cache coherency scales, and it is possible to have shared memory architecture with thousands of cores.

Acknowledgements

I would like to take this opportunity to express my deepest thanks to my first supervisor, Dr. Wim Vanderbauwhede, for his patient and excellent advice throughout this study and for his encouragement, guidance, and support over the past four years, which enabled me to be a better researcher. I would also like to extend my gratitude to my second supervisor, Dr. Lewis Mackenzie, for his unwavering belief in me and his support and understanding throughout my PhD study.

I would like to acknowledge and thank His Majesty Sultan Qaboos and the government of Sultanate Oman for funding this research. I would like to acknowledge the German University of Technology in Oman for giving me a study leave to do this research. I would like to express my appreciation to all staff at the School of Computing Science for their help and support during these years.

I take this opportunity to express my profound gratitude to my beloved parents, wonderful siblings, and sincere friends for their love, moral support, and patience during my study. Lastly, I offer my regards and blessings to all of those who supported me in any aspect during these years.

To my beloved family and wonderful friends.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Shift Towards Many-Core Systems

The scaling of semiconductor technology has enabled billions of transistors to be integrated into a single chip, following Moore's Law that the number of transistors on a single chip doubles every two years. There are three main limitations in the scaling of a single core:

- A power wall is due to the limitation on the scaling of clock speeds and because the ability to handle on-chip heat has reached its physical limit.
- A memory wall occurs because the gap between memory access and processing speed is growing further. In addition to improving the average memory access, caches are getting bigger.
- A instruction-level parallelism (ILP) wall arises from the super-linear increase in execution unit complexity without linear speedup in application performance.

Multi-Core Processors (MCP) were the solution to overcoming the limitations of the single-core processor, moving from sequential computing to parallel computing. This solution allowes for the continuation of Moore's Law. However, MCP came with its own limitations, one of which was the imperfect scaling as the performance was dependent on serial code. Another limitation was the difficulty of optimising a software to run in parallel. It is easier to add more cores than for the software to take advantage of a multiple-core processor. The end of Moore's law is widely expected in the coming decade. The many-core system is one of the solutions. The definition of many-core is a multi-core architecture with a particularly high number of cores.

1.2 Motivation and Objectives

There are a number of issues that need to be considered when building a many-core system:

- Power
- Connectivity
- Arbitration
- Memory Issues
- Cache Coherence
- Scheduling
- Programmability

In this research, we focus on three of the issues that many-core systems need to address: connectivity, memory issues, and cache coherence. A massive many-core system requires high performance interconnections to transfer data between the cores on the chip. For many-core processors with close to one hundred cores, such as the Tilera Tile64 [2] or the Intel MIC [3], Network-on-Chip (NoC) has become the preferred on-chip communication infrastructure. Performance of NoC based many-core systems is highly dependent on the traffic patterns and the NoC topologies. In many-core systems, communication, not computation, is typically the performance limiting factor. In ten years, according to the International Technology Roadmap for Semiconductors (ITRS), we can expect that a thousand-core processor can fit into an area of less than $1\,mm^2$ and it will consume only a few watts.

The NoC architecture of such many-core processors for exascale systems is of particular interest. Besides the standard mesh, as used in the Tilera many-core system and the Intel SCC, and the ring topology, as used in the Intel Xeon Phi and recent Intel Xeons [3], many NoC topologies have been proposed and evaluated. Some aimed to provide improvements on the ring topology, such as the Spidergon [4] or the Quarc [5]; some aimed to improve on the mesh, e.g. the concentrated mesh [6]. There has also been interest in scale-invariant topologies such as the quadtree [7].

With regards to memory issues in many-core systems, 3D memory stacking has been the focus of research in recent years [8–11], since it enables much higher memory bandwidth and mitigates the memory wall problem of 2D integration. The latest Xeon Phi (Knights Landing) already uses a hybrid form of stacked memory, Micron's Hybrid Memory Cube [12]. Further integration leading to 3D memory stacked on top of the processor is in active research. 3D memory allows each processor core to have fast and high bandwidth access to the memory directly stacked on top of it using very dense vertical interconnections [13–15]. Some of the advantages of 3D memory stacking are lowering the latency and increasing the memory bandwidth.

Cache coherency is important when many cores are sharing data. Cache coherency ensures the consistency of shared data that is stored in multiple memory locations. The conventional wisdom has been that cache coherence could not scale because of exploding storage and interconnection network traffic requirements, as well as concerns over latency and energy consumption. However, some researchers have suggested techniques that enable cache coherency to scale.

Locality of computations might be a solution for increasing the performance of many-core systems. The performance of future many-core systems with thousands of cores will heavily depend on the Network-on-Chip architecture to provide scalable communication. Communication locality is essential to reduce latency and increase performance. As the number of cores increases, locality will only become more important. Many-core systems should be designed so that cores communicate mainly with the neighbouring cores, to minimise the communication cost.

1.3 Thesis Statement

The thesis statement is established as follows:

Memory architectures that support locality of computations on many-core Network-on-Chip (NoC) communication subsystems can improve the performance in terms of latency and throughput.

Distributed memory architecture and shared memory architecture can scale and improve their communication performance if locality of computations is introduced. Many-core systems that force cores to communicate mainly with their neighbouring cores can minimise the communication cost. Locality of computations can increase bandwidth and reduce latency. The ITRS physical data shows that the wire delay will increase significantly in the future; therefore, the locality of computations will be crucial towards increasing many-core NoC performance.

1.4 Thesis Contributions

The main findings from this work are summarised as follows:

- We establish that the link complexity, routers, buffers, area space, and wire space overhead of the NoC on a thousand-core system are negligible, and the choice of topology depends solely on the performance in terms of latency and throughput.
- In distributed memory architecture, locality of computations on many-core NoC communication subsystems improves the system performance in terms of latency and throughput. Scale-invariant topologies such as the fat quadtree perform worse than flat ones (esp. cmesh).
- In shared memory architecture, data locality on 3D stacked hierarchical cache architecture for many-core NoC communication subsystems without cache coherency traffic improves the system performance in terms of latency and throughput. Scale-invariant topologies such as the fat quadtree perform better than flat ones.
- Cache coherency scales using hierarchical cache architecture and locality of computations, with only slight traffic overhead and slight overhead of the directory on the cache size.

1.5 Overview of the Thesis

The remainder of this thesis is organised as follows:

Chapter 2: Background and Related Work

This chapter presents general background information about Network-on-Chip (NoC) and its characteristics in terms of topology, routing algorithms, and switching techniques. The chapter delivers an overview of memory architectures, which are distributed memory, shared memory, and distributed shared memory. In addition, it summarises the parallel programming models and provides a brief overview of some of the most common programming models: MPI, openMP, and PGAS. Moreover, it explains many-core systems and gives a number of examples of commercialised many-core platforms that exist nowadays.

Chapter 3: Cost Model, Technology Node Assumptions, and Methodology

This chapter describes the cost models in a many-core system for the link complexity, the routers, the buffers, and the area space for different topologies. It provides the technology node assumptions for realistic future many-core system parameters, which we made using data from the International Technology Roadmap for Semiconductors (ITRS) for the year 2023. This chapter compares a number of Network-on-Chip simulation tools, explains in detail how the Heterogeneous Network-on-Chip Simulator (HNOCS) works, and gives reasons as to why we have selected it as a simulation tool for this research. HNOCS-XT is our extended version of HNOCS because our research has some requirements that are not supported in HNOCS. This chapter provides the experimental methodology we use in this research to measure the performance of different topologies by enforcing locality. We use group clustering and ring clustering models to select the neighbouring cores communicating with each other based on the locality parameter.

Chapter 4: Effects of Locality on Message Passing Communication Performance

This chapter describes the effects of communication locality on many-core Message Passing Model (MPM) performance. In distributed memory architecture, cores communicate with each other in NoC by passing messages. Using communication locality on a many-core system can improve its performance as programs run on a large many-core system can benefit from local communication, where data are placed in nearby cores. Thus, in this chapter, we show the significance of locality in the future many-core systems and that our locality model expresses Rent's rule. This chapter provides a brief description of the topologies used in the simulations. We compare the topologies in terms of latency and throughput. Furthermore, the chapter details the simulation setup, the results, and discusses the many-core NoC simulation results. We show that communication locality improves many-core NoC performance and that the increase in NoC performance depends on the network topology. The following papers are published based on the results of this chapter:

- "Evaluation of the Memory Communication Traffic in a Hierarchical Cache Model for Massively-Manycore Processors." 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). IEEE, 2016.
- "The Performance of NoCs for Very Large Manycore Systems under Locality-based Traffic." International Journal of Computing and Digital Systems. Volume : 5. Issue: 2. Issue Publication Date: March 2016.

Chapter 5: Effects of Locality on Shared Memory Communication Performance

This chapter presents the effects of data locality on a shared memory model in many-core systems. Data locality is very significant in improving many-core performance, as applications mapped on a large many-core system can take advantage of short distance communication, where data are placed in cache banks closer to the cores accessing them. Therefore, in this chapter, we design a hierarchical cache model that supports data locality. This chapter shows the hierarchical placement of caches in different topologies on the proposed hierarchical cache model. In addition, it details the parameters used for the simulations and compares the performance of the many-core architecture in terms of latency and bandwidth using the ITRS physical data for 2023. The results show that the model of thread placement and the distances at which they are placed significantly affect the NoC performance. Furthermore, they confirm that scale-invariant topologies perform better than flat topologies. The following paper is published based on the results of this chapter:

- "The Performance of NoCs for Very Large Manycore Systems under Locality-based Traffic." International Journal of Computing and Digital Systems. Volume : 5. Issue: 2. Issue Publication Date: March 2016.

Chapter 6: Cache Coherency for Many-Core Systems

This chapter presents the cache coherency protocol we suggest for many-core systems. It explains the hierarchical tracking of shares. This chapter provides the cache coherency protocol overhead on the cache size and on traffic in the shared memory hierarchical cache model. It details the simulation setup, the results, and discusses the simulation results. The chapter shows that cache coherency scales, and its effect on the system performance is low when using the hierarchical cache model.

Chapter 7: Conclusion and Future Work

Lastly, in this chapter, we present the thesis conclusions and suggest some directions for future many-core systems.

Chapter 2

Background and Related Work

This chapter presents background information about Network-on-Chip (NoC) and the characteristics of NoCs in terms of topology, routing algorithms, and switching techniques. The chapter also provides an overview of memory architectures, which are distributed memory, shared memory, and distributed shared memory. Moreover, it describes the many-core systems and gives some examples of many-core platforms that are commercially available nowadays.

2.1 Network-on-Chip

The concept of Network-on-Chip (NoC) originated as a new on-chip communication approach used to design the communication subsystem between Intellectual Property (IP) cores in a System-on-a-Chip (SoC). It offers better scalability, throughput, and overall latency compared to a bus-based on-chip interconnect. In designing NoC, the most essential characteristics are network topology and routing algorithm, along with congestion control methodologies, link capacities, arbitration methods, number of buffers, and virtual channels per link, etc. Moreover, the research area of NoC is still in its early stages. Therefore, new architectures, techniques, and ideas are being proposed, developed, and evaluated [16], [17].

A Network-on-Chip is composed of three main building blocks: routers, links, and network adapters. The links and the routers implement the functionality of physical, data-link, and network layers of the OSI reference model [18]. Each router has a set of ports, which are used to connect the router to the network. The routers implement the routing algorithm, and switching technique. The links connect between the routers. They may consist of one or more logical or physical channels [19].

The network adapters implement the interface by which cores connect to the NoC. Their function is to decouple computation of the cores from communication of the network. In most NoC architectures the network interface implements the functionality of the transport and session layers of the OSI reference model [17], [19].

An Intellectual Property (IP) core can be any computation or storage component that has been employed in traditional SoC development, including processor, memory, FPGA, or DSP. A core may be comprised of a number of IP cores that are connected to each other by any communication architecture, including a NoC [17], [19].

2.1.1 Topology

The topology of a NoC identifies as the physical interconnection network. It describes how nodes, switches, and links are connected to each other. Topologies for NoCs can be classified into two categories: direct network topologies, where each node is connected to at least one core, and indirect network topologies, which have a subset of nodes not connected to any core and performing only network operation. Many on-chip network topologies are found in the literature, such as fat-tree [7], mesh [20], torus [21], hexagon [22], spidergon [23], quarc [5], and butterfly [24]. Implementing a particular topology for a NoC is vital since the result is a trade-off between metrics such as performance, energy, and cost. The topology of a network affects the scalability, performance, complexity of the routing algorithm, and power consumption.

One of the main properties of a topology is bisection bandwidth. The bisection width is the number of wires that must be cut when the network is divided into two equal sets of nodes, and the bisection bandwidth is the collective bandwidth over these wires. Bandwidth refers to the maximum capacity of data transmission over a communication network [20]. System performance increases when the bandwidth is higher. As more nodes are attached to the network, a larger volume of communication and more bandwidth are required. If the network bandwidth does not scale appropriately with the number of nodes, excessive traffic will lead to high message latency and decreased performance. However, networks with high bisection bandwidth will require more routers and more wires per node, which consume considerable space and increase the cost of the system [20].

Mesh Topology

The mesh topology is the simplest and most popular NoC topology so far due to its regular structure with uniform router design, easy scalability, and high compatibility with standard silicon fabrication technologies [25]. It is used in most of the recent many-core chips, such as Intel SCC 48-core [26], TFlops 80-core [27], and Tilera 64-core [2].

The mesh topology consists of a grid of routers with interconnecting cores placed along with the routers. Every router, except those at the edges, is connected to four neighboring

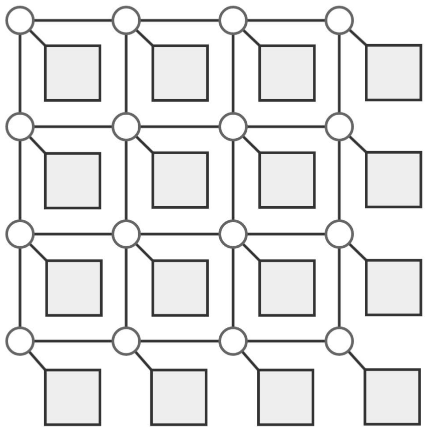


Figure 2.1: Mesh

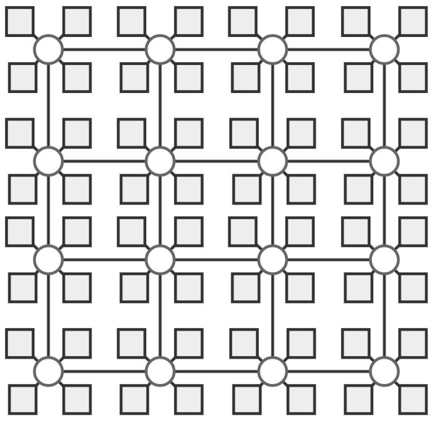


Figure 2.2: Concentrated mesh

routers and one core. Therefore, the mesh has a radix (number of ports) of five. In a mesh, the number of routers is equal to the number of cores. Addresses of routers and cores can be easily defined as x-y coordinates in a mesh. Figure 2.1 shows the layout for a mesh of 16 nodes. The mesh topology assumes that all the links have the same length. It is easier to predict the area requirements for the mesh topologies, because area requirement grows almost linearly with an increase in the number of cores. In a mesh network, deadlock is avoided by using a deadlock-free routing algorithm such as an XY routing algorithm [28].

The mesh topology has a relatively high network diameter, which is a drawback; hence, the concentrated mesh topology was introduced.

Concentrated Mesh Topology

The concentrated mesh (cmesh) was introduced by Balfour and Dally [6] to preserve the advantages of a mesh with decreased diameter. The number of cores sharing a router is called the concentration degree of the network. In this work we use a degree of four. Therefore, in

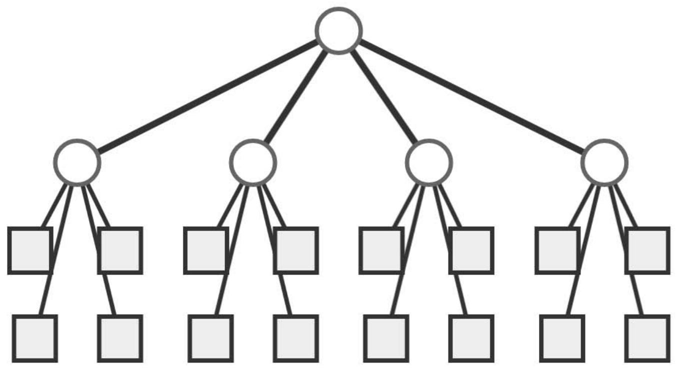


Figure 2.3: Fat quadtree layout

the cmesh topology a set of four neighboring cores is connected to the same router. It scales better than the 2D mesh topology because of its lower hop count and consequently improved latency. It has a radix of eight. Despite the larger number of ports, having a quarter of the number of routers in 2D mesh leads to large savings in power consumption [29]. Figure 2.2 shows the layout for a concentrated mesh of 64 nodes. Routing in cmesh is using XY routing in the mesh structure and once the flits arrive to the parent router of the core, the router sends the flit to the appropriate core.

Fat Quadtree Topology

The tree topology interconnects the routers in a tree form with the cores at the leaves. A major disadvantage of a tree network is its bisection width. Hence, in order to overcome this drawback and to avoid congestion towards the root of the tree, a fat tree is introduced. A fat tree uses an increasing number of point-to-point links per connection, as described in [7]. The number of links is multiplied by the tree degree as we move toward the root. A fat quadtree (FQT) of size N is a structure that can be regarded as a rooted 4-ary tree of height $log_4(N)$. Figure 2.3 shows the layout for a fat quadtree of 16 nodes. The fat quadtree of size N has $(4N-1)/3$ routers. The advantage of the fat quadtree over the mesh is that the communication diameter of a fat quadtree is only $O(log_4 N)$ compared to $O(\sqrt{N})$ for the mesh. The fat quadtree uses the nearest-common ancestor routing algorithm. Packets are routed adaptively up to the common ancestor and deterministically down to the destination core. There are no cyclic connections in the fat quadtree; hence, it is deadlock-free.

Ring Topology

The ring topology is a popular topology for many communications and parallel processing applications. This is due to its structural simplicity and very simple routing protocols. The routers connect in a ring structure. Every router is connected to two neighbouring routers

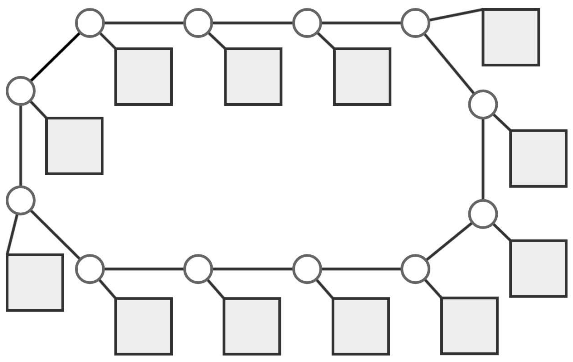


Figure 2.4: Ring layout

and one core [30]. Hence, the ring has a radix of three. In a ring topology, the number of routers is equal to the number of cores. Figure 2.4 shows the layout for a ring of 12 nodes. The ring topology is used in chips, including the IBM Cell processor [31], the Intel Larrabee processor [32], and the Intel Xeon Phi [33].

2.1.2 Switching Techniques

Switching techniques are determined by how the input channel of the switch is connected to the output channel, selected by the routing algorithm, and when the data can transfer along the channels. Data are transmitted in the network as messages that are divided into packets, which in turn are divided into flow control units (flits). Switching technique typically has a more significant impact on performance than routing and topology [20]. Interconnection network switching can be circuit switching, store-and-forward switching, virtual cut-through, or wormhole switching.

Circuit Switching

Circuit switching reserves the circuit from source to destination from the time the connection is established until the transport of data is complete [20, 34]. This is done by injecting the routing head-flit into the network. The head-flit contains the source and destination addresses and some additional control information. While the head-flit is transmitted to the destination through intermediate routers, it reserves the physical links. Once it reaches the destination, an acknowledgment is transmitted back to the source. Then, the message content can be transmitted to the destination. The reserved physical links are released by the destination or by the last few bits of the message.

Store-and-Forward Switching

Store-and-forward switching is a packet-switched protocol in which the routers store the complete packet and forward it based on the information within the header. In other words, each packet is individually routed from source to destination. A packet is buffered at each intermediate router before it is forwarded to the next router. Therefore, the packet may reside in an intermediate router if the next router does not have sufficient buffer space [20, 34].

Virtual Cut-Through Switching

Virtual cut-through switching has a forwarding mechanism similar to the wormhole switching. However, before forwarding the first flit of the packet, the router waits for a guarantee that the next router in the path will accept the entire packet. In case there are no guarantees, the router must be able to store the whole packet [20, 34].

Wormhole Switching

Wormhole switching is the most popular technique used on-chip [17]. It combines packet switching with the data streaming quality of circuit switching to achieve minimal packet latency. The router looks at the routing information in the head-flit to determine its next hop and immediately forwards it. The following flits are pipelined through the network following the head-flit. This causes the packet to worm its way through the network, possibly bridging a number of routers. Each intermediate router has enough space for a few flits, which means less buffer size is required than in store-and-forward switching. A delayed packet, however, has the unpleasantly expensive side effect of occupying all the links that the worm spans [20, 34].

2.1.3 Routing

Routing algorithms decide the path from source to destination. The effect of routing algorithms is significant to the network topology. Numerous routing algorithms have been proposed and adopted for interconnection networks. Nevertheless, the uniqueness of Network-on-Chip prevents employing most of the contributions. Routing tables and complex arbitration protocols must be avoided for on-chip networks, and efficient routing algorithms should be used to target small area and high frequency implementation [5]. Deterministic routing, source routing, and adaptive routing are mainly used for NoC connections.

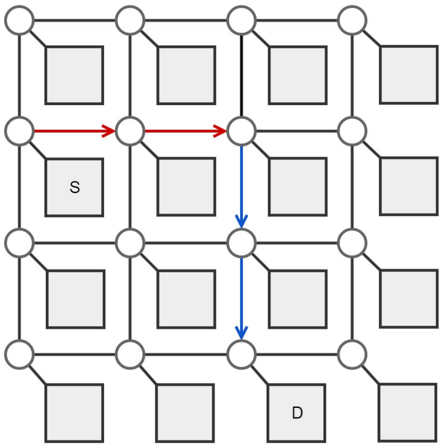


Figure 2.5: Example of XY routing

Deterministic Routing

In deterministic routing, the path is determined by the source and destination alone. The path between source and destination is always the same. Each intermediate router computes the next hop. It requires only the destination address to decide the next hop. A popular deterministic routing scheme for NoC is XY routing [17], [35]. Figure 2.5 shows an example of XY routing. In a 2-Dimesion mesh topology, each router can be identified by its coordinate (x, y). The XY routing is when the packects are routed along the X-axis then Y-axis to reach its destination [28].

Source Routing

In source routing, the source core specifies the entire path to the destination. A path from the source to the destination is stored in the packet header. The path can be an ordered list of addresses of all the intermediate routers. In the intermediate router, the path is typically modified in order to represent the appropriate routing choice for the next router [17]. Figure 2.6 shows an example of source routing.

Adaptive Routing

In adaptive routing, the routing path is decided on a per hop basis. At each intermediate router, the decision is not based on the destination address only. The traffic information is taken into account as well. This results in more complex router implementations but offers benefits like dynamic load balancing [17], [20].

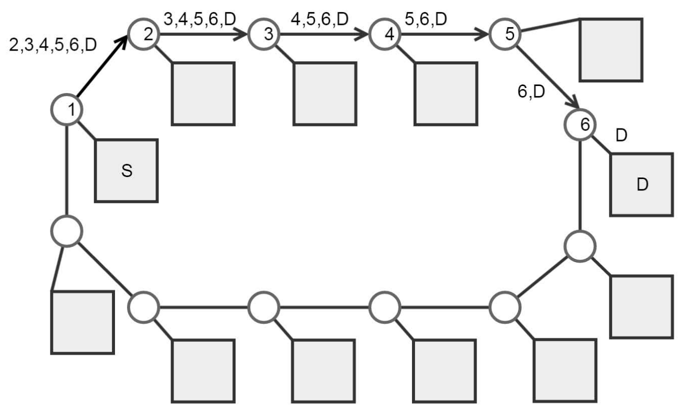


Figure 2.6: Example of source routing

2.2 Memory Hierarchy

Memory is important to the operation of a computer system. Memory refers to any physical device capable of storing information temporarily or permanently. The concept of memory hierarchy is more important to the development of the modern memory system than it was in early systems [36]. Memory hierarchy is usually illustrated as a pyramid, as shown in Figure 2.7. The higher levels in the pyramid have better performance than the lower levels in the hierarchy. Performance in the memory hierarchy is understood in term of response time, complexity, and capacity. The purpose of the memory hierarchy is to allow the CPU to have faster access to data, so that the CPU can spend more time working, instead of waiting for data. A modern computer system moves data that are needed from the slower memory up to the faster memory. This is why the small and expensive fast memory is located near the CPU, so that the processor can quickly access the data needed [37]. Computer memory can be divided into two groups: primary memory and secondary memory [38].

2.2.1 Primary Memory (Volatile Memory)

The primary memory consists of fast memory which is directly accessible by the CPU. This type of memory is volatile. Volatile memory is a memory that loses its content when the computer or hardware device loses power, in other words, it cannot save its state unless it is powered on. Consequently, when a computer system is power off or rebooted, all data in the primary storage is lost. Primary storage is fast, typically small and expensive [36, 37]. CPU registers, caches and random access memory are some examples of primary memory.

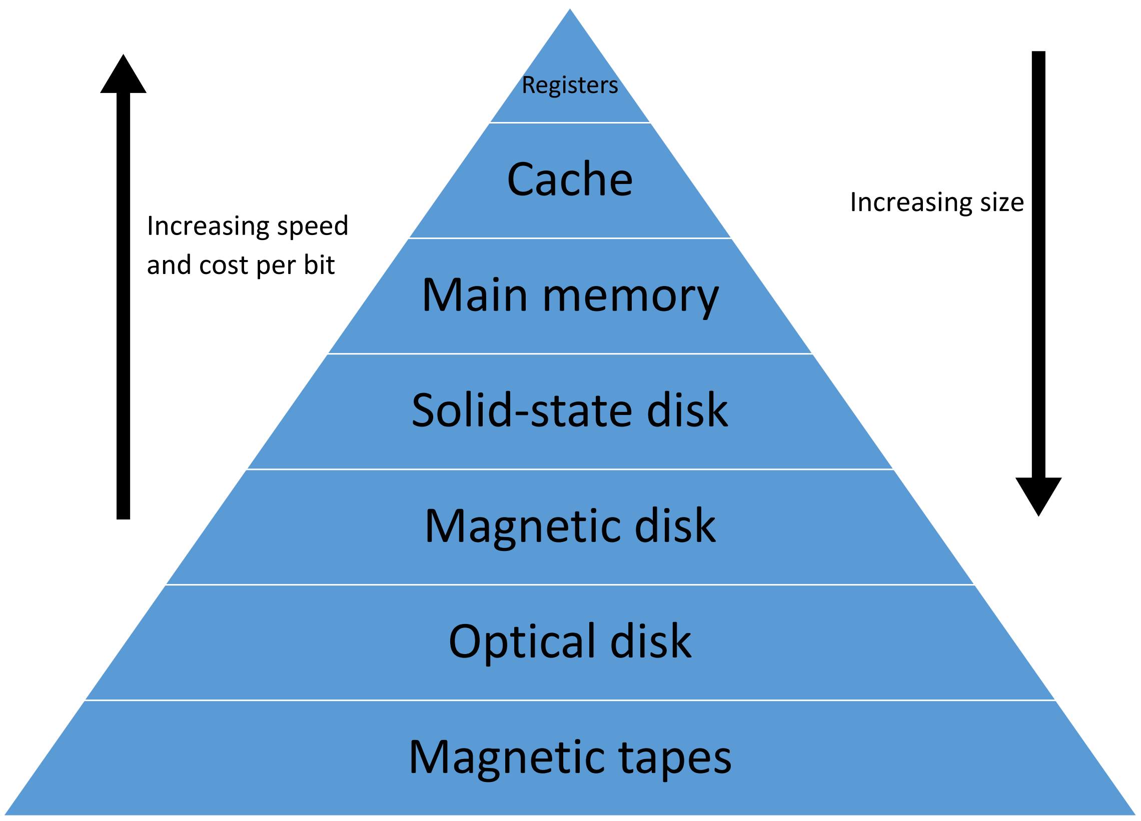


Figure 2.7: Memory hierarchy

CPU Registers

CPU registers are at the very top of the memory hierarchy. There are only a few processor registers in a CPU, but they are very fast. A computer moves data from other memory into the registers, where it is used for different computations [36].

Caches

Cache memory is a fast memory which is located between the main memory and the CPU registers in the memory hierarchy. Cache memory stores most recently accessed data and instructions, to allow future accesses by the processor to particular data and instructions to be handled much faster. Caches have a hierarchy of their own. Most computers nowadays have three levels of cache [39, 40]. IBM zEnterprise 196 (z196) system uses four levels of cache [41].

Main Memory

Main memory comes next in the hierarchy after caches. The main memory refers to Dynamic Random-Access Memory (DRAM) which means it is relatively cheap, but still quite fast. It performs both read and write operations on memory. If power failures happen in systems during memory access then you will lose your data permanently because RAM is volatile memory [37].

2.2.2 Secondary Memory (Non-Volatile Memory)

Secondary storage is in many ways the opposite to primary storage. It is a non-volatile memory which means it can save its state even when powered off. Accordingly, when a computer system is powered off or rebooted all the data stored in the secondary storage is saved and accessible when the system is next time started. Secondary storage is generally named external memory and it is characterized by being large, slow, and cheap. Secondary memory can not be accessed by the CPU directly, which makes it even more complicated and slow to access than primary memory [36].

2.3 Memory Architectures

Memory architecture describes the design used to build the data storage in a way that is a combination of the fastest, most reliable, most resilient, and least expensive way to store and retrieve information. As the number of cores in a chip increases, the latency on-chip interconnects take a longer time. On-chip interconnects refer to the exchange of data between memory and cores as well as the exchange of data between cores. Therefore, depending on the specific application, a trade-off in either the latency or the throughput may be necessary in order to improve another requirement. The main models of memory architectures are: distributed memory architectures, shared memory architectures, and distributed shared memory architectures.

2.3.1 Distributed Memory Architectures

The memory is physically distributed among processors and each processor has its own private memory, which cannot be accessed by other processors. A processor can only access its own memory. Access to data belonging to the memory of another processor must be explicitly requested; therefore, the communication in this type of architecture occurs by message passing [42]. In distributed memory, memory is scalable with the number of processors. The number of processors increases proportionately to the size of memory. Each processor can quickly access its own memory without the overhead of cache coherency. The disadvantage of distributed memory is that the programmer is responsible for many of the details related to data communication between processors. In addition, it may be difficult to map existing data

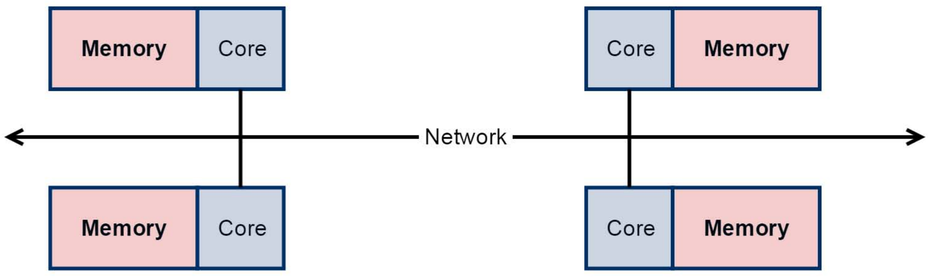


Figure 2.8: Distributed memory architectures

structures, based on global memory, to this memory organisation [43]. Figure 2.8 illustrates the distributed memory architecture.

2.3.2 Shared Memory Architectures

All memory is mapped onto a global memory address space, which is accessible by all processors in the system. In shared memory, communication is performed by writing to and reading from shared variables. The shared variables data is visible to all processors in the system. Therefore, managing concurrent access to the shared variables is important. As the number of processors grows in a shared memory machine, it becomes harder to guarantee that every processor has fast access to the shared memory [42].

Symmetric Multiprocessor Architecture

Symmetric Multiprocessors (SMP) is a case of shared memory. In SMP, all processors have the same access to any memory location [44]. Because SMP has a uniform memory latency for all processors, it is also known as Uniform Memory Access (UMA). SMP has a centralised shared main memory operating under a single operating system with two or more processors. Usually, each processor has an associated private cache memory to speed up the main memory data access and to reduce the system bus traffic. This architecture is not really scalable because it uses a central bus, which provides a constant bandwidth shared by all processors [45]. Figure 2.9 illustrates the symmetric multiprocessor architecture.

Non-Uniform Memory Access Architecture

Non-Uniform Memory Access (NUMA) is a case of shared memory as well; however, processors have faster access to their own local memory. In NUMA, processors are connected to a scalable network, and each processor has a portion of memory attached directly to it. The primary difference between NUMA and distributed memory is that in NUMA processors can

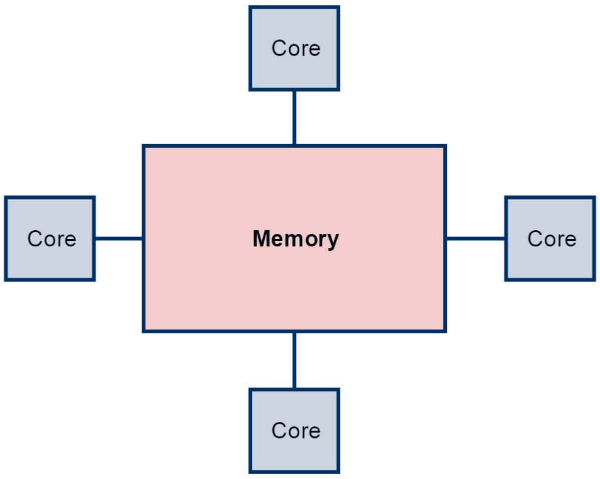


Figure 2.9: Symmetric multiprocessor architecture

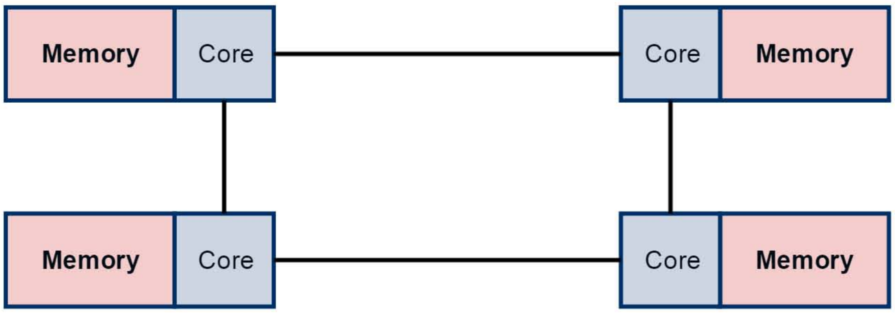


Figure 2.10: Non-uniform memory access architecture

have mappings to memory connected to other processors; however, this is not the case for distributed memory, as a processor does not have mappings to memory connected to other processors [45]. Figure 2.10 illustrates the NUMA architecture.

Non-Uniform Cache Architecture

In the traditional cache architectures, each level in the cache hierarchy has a single and uniform cache access time. The increasing communication delay causes the successful access time to large on-chip caches to be a function of a line's physical location within the cache. Accordingly, cache access time becomes a range of latencies rather than a single discrete latency. However, non-uniform access to caches can provide access to portions of the cache that are closer to the core [46]. Non-Uniform Cache Architecture (NUCA) is a design paradigm for large on-chip caches and was first proposed by Kim et al. [47]. It has been introduced to deliver lower access latencies. The NUCA systems are built by dividing the entire cache into smaller banks. Each of these banks has a single discrete latency [48].

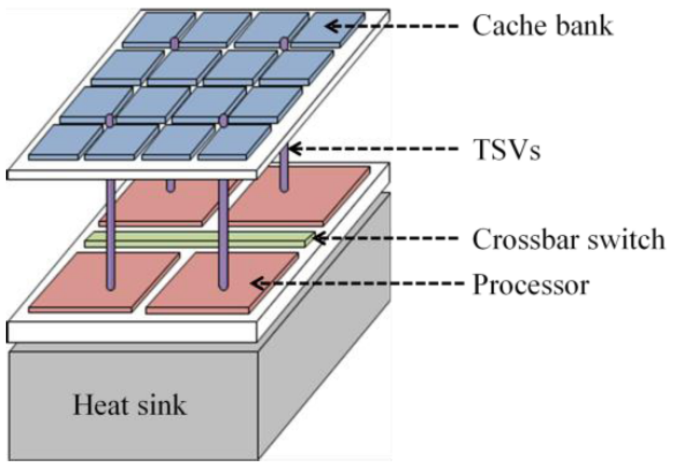


Figure 2.11: Example of 3D memory stacking architectures (from [1])

2.3.3 Distributed Shared Memory Architectures

Distributed Shared Memory (DSM) is a form of memory architecture where the physically distributed memory can be addressed as one logically shared global address space. Most of the supercomputers that exist today have both shared and distributed memory architectures. Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future. This architecture is scalable; however, it has increased programming complexity [49].

2.3.4 3D Memory Stacking Architectures

3D memory stacking has received great attention in recent years, since it resolves the memory bandwidth challenges of 2D integration [13–15]. 3D memory stacking involves packaging the memory on top of the processor cores, which allows each processor core to have fast and high bandwidth access to the cache banks directly stacked on top of it, using very dense vertical interconnects. The processor core can also address cache banks stacked on the other processors [13, 14]. The advantages of 3D stacking are lower latency and higher bandwidth. The proposals of the 3D stacked cache memory hierarchy have similar concepts to the memory hierarchies of the traditional multiprocessors. For example, the stacked cache can be either private or shared [1]. Figure 2.11 shows an example of 3D memory stacking architecture.

2.4 Cache Coherence

Cache coherence is a concern in a multi-core environment because of the distributed local caches. Since each core has its own cache, the copy of the data in that cache may not always be the most updated version. Cache coherency can be understood as the mechanism that ensures the consistency of data stored in local caches of a shared resource.

The easiest and safest way to do this is to update all copies as soon as any processor writes on them. However, this leads to poor parallel processing performance because all processors have to stall until their copies are updated. As a result, relaxed models exist in which the copy update might be delayed. Cache coherency is a property of an individual memory location while memory consistency refers to the order of accesses to all memory locations. Memory consistency guarantees that the correct result of the program will not be affected by delays in the memory updates. Many consistency models exist where the cache copies must be updated, such as sequential and processor consistency. The implementation of memory consistency is called synchronization. In shared memory systems, synchronization can be implemented implicitly using cache coherence.

Many researchers think that cache coherency will not scale to a large number of cores [18], [50], [51]. However, Martin and others [52] show that it is possible, using a combination of known techniques for scaling cache coherence. Shared caches augmented to track cached copies, explicit cache eviction notifications and hierarchical designs are some of these techniques.

There are two basic ways of implementing cache coherence, namely the snooping and the directory-based protocols.

2.4.1 Snooping Protocol

Snooping cache coherency protocols can only be used in systems where multiple processors are connected via a shared bus [53]. Hence, snooping protocol takes advantage of the bus by using the broadcast method. Snooping protocols mainly have two different ways to implement coherency: write-invalidate and write-broadcast. In write-invalidate, a processor sends invalidation messages to all other processors that have cached copies of the shared data, and then it updates its own copy. In write-broadcast, a processor broadcasts the updates made to shared data to other processors who have a cached copy, so that all copies of the shared data are the same. Processors can read the shared data without any coherence problems; however, a processor must have exclusive access to the bus in order to write. MESI protocol [54] and MOESI protocol [55] are some examples of snooping cache coherency protocols.

Snooping protocols are not scalable because they require a shared bus connection, which limits the number of processors that can be attached to the bus. In addition, snooping protocols use broadcasting techniques that limit the performance due to competition for shared resources. Consequently, they cannot be used in many-core NoC.

2.4.2 Directory-Based Protocol

Directory-based cache coherence protocols were introduced to deal with cache coherence in systems containing more processors than the bus can accommodate [56]. In a directory-based system, the shared data is placed in a common directory that maintains the coherence between caches. The processors must ask permission from the directory to load shared data from the primary memory to its cache. When the shared data are modified, the directory sends either updates or invalidations to the other caches that have copies of the shared data. The directory can be placed centrally with the main memory in a multi-processor system or can be distributed among the caches. DASH coherence protocol [57] and SGI Origin coherence protocol [58] are some examples of directory-based cache coherence protocols.

The directory-based protocols are scalable to a large number of processors. There are two challenges of implementing directory-based cache coherence protocol, which are reducing the overhead of directory storage and reducing the number of messages required to implement coherence protocol.

2.5 Many-Core Systems Processors

Many-core system processors are processors containing a large number of simpler, independent pro- cessor cores. They are designed for a high degree of parallel processing. There is much research about many-core design methods [59], [60], [61], which shows that many-cores will continue to be integrated into chips in the future. This section describes a number of existing many-core processors.

2.5.1 Xeon Phi Processor

Xeon Phi is manufactured by Intel. It has 60 cores connected by a bidirectional ring interconnect [33]. Each core has a $22\,nm$ processor, 4-way hyper-threading, $32\,KB$ L1 private cache and $512\,KB$ L2 private cache. The processor runs at a speed of $1.053\,GHz$. The memory bandwidth is $320\,GB/s$. The Xeon Phi has eight memory controllers. Each direction in the bidirectional ring interconnect is comprised of three independent rings. The first ring is the data block ring. The data block ring is $64\,B$ wide to support the high bandwidth requirement due to the large number of cores. The second ring is the address ring, which is used to send

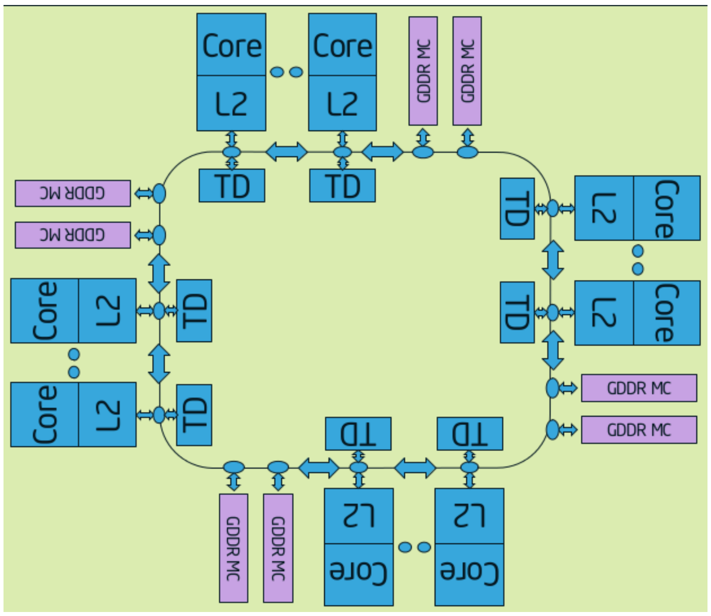


Figure 2.12: Xeon Phi processor architecture

read/write commands and memory addresses. The third ring is the acknowledgment ring, which sends flow control and coherence messages [62, 63]. Figure 2.12 shows the Xeon Phi processor architecture.

Cache coherence is maintained using a global-distributed tag directory-based protocol. When a cache miss occurs in a core's L2 cache, an address request is sent on the address ring to the tag directories. The memory addresses are uniformly distributed amongst the tag directories on the ring. If the requested data block is found in another core's L2 cache, a forwarding request is sent to that core's L2 over the address ring, and the request block is subsequently forwarded on the data block ring. If the requested data is not found in any caches, a memory address is sent from the tag directory to the memory controller.

2.5.2 TILEPro64 Processor

TILEPro64 is a multi-core processor originally manufactured by Tilera [64]. It has 64 cores in an 8×8 mesh network with cache coherency. Each tile has a general purpose $90\,nm$ processor, cache, and a router. The processor runs at a speed of $866\,MHz$. The memory bandwidth is $200\,GB/s$. TILEPro64 has four memory controllers. Each core has a private $8\,KB$ L1 data cache, a private $16\,KB$ L1 instruction cache, and a private $64\,KB$ L2 cache. The combination of all L2 caches across the chip form the distributed L3 cache [2,65]. Figure 2.13 shows the TILEPro64 processor architecture.

Cache coherence is maintained using in-cache directory-based protocol. When a miss occurs in a local L2 cache, the core sends a request to the core that has the cache line in its

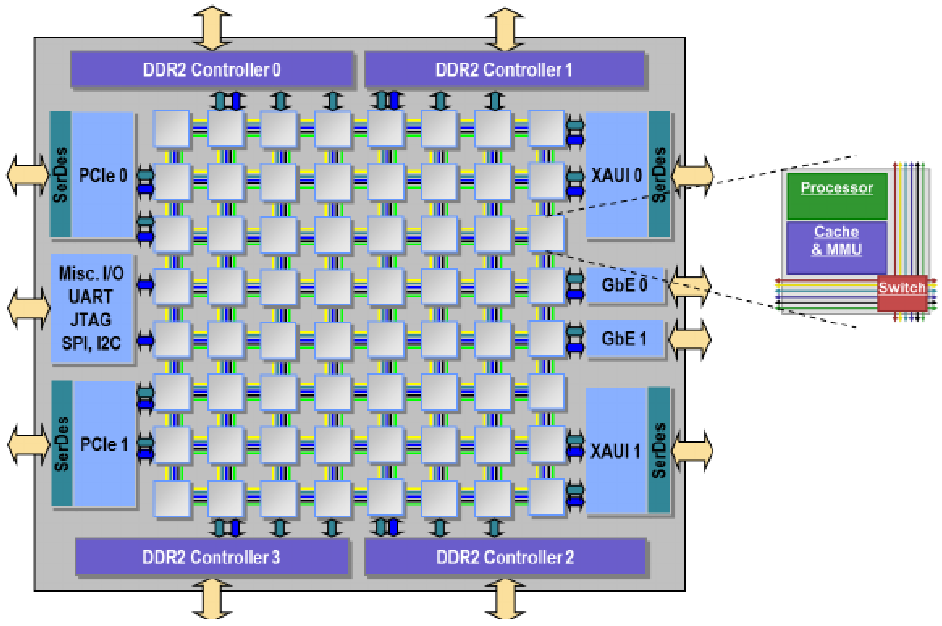


Figure 2.13: TILEPro64 processor architecture

private L2 cache read and copy the cache line into its own local L2 cache. TILEPro64 has six interconnected meshes. Three of the networks are dedicated for hardware control and managing memory movement and cache coherence, in order to speed the cache coherence communication across the chip. The other three networks are allocated to software.

2.5.3 SW26010 Processor

SW26010 is a 260 core many-core processor designed by the National High Performance Integrated Circuit Design Center. The processor frequency is $1.45\,GHz$. It has a memory bandwidth of $136.51\,GB/s$. The SW26010 has four clusters that contain one management processing element (MPE), one memory controller (MC), and 64 Compute-Processing Elements (CPEs), which are arranged in an 8×8 mesh. The four clusters are connected by Network-on-Chip. Each MPE has a $32\,KB$ L1 instruction cache, a $32\,KB$ L1 data cache, and a $256\,KB$ L2 cache [66]. Figure 2.14 shows the SW26010 processor architecture.

2.5.4 PEZY-SC Processor

PEZY-SC (SC: Super Computing) is a many-core processor developed by Pezy Computing. PEZY-SC has 1024 cores and each core has a $28\,nm$ processor. The processor runs at a speed of $733\,MHz$. The memory bandwidth is $1533.6\,GB/s$. PEZY-SC has a $1\,MB$ L1 cache, a $4\,MB$ L2 cache and an $8\,MB$ L3 cache [67]. Figure 2.15 shows the PEZY-SC processor architecture.

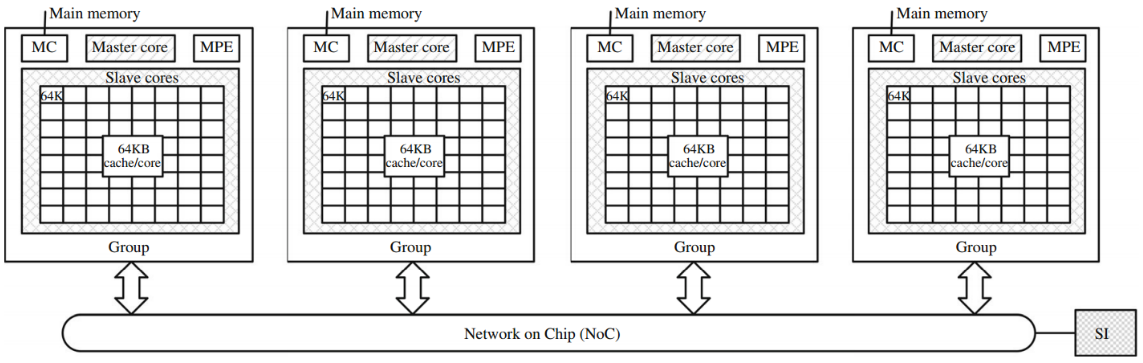


Figure 2.14: SW26010 processor architecture

The cores are classified into three layers: a village, which is 4 cores, each with a private L1 cache, a city, which is 4 villages, each with a L2 cache, a prefecture, which is 16 cities, each with a L3 cache, and 4 prefectures. Cache coherency is not integrated in the processor [68].

2.6 Summary

In this chapter, we have discussed Network-on-Chip (NoC) in general and the important aspects of NoCs in terms of topology, routing algorithms, and switching techniques. We have provided an overview of the different types of memory architectures: distributed memory architectures, shared memory architectures, and distributed shared memory architectures. In addition, we have explained programming parallel models. We have provided a brief detail of some common programming models: MPI, openMP, and PGAS. Moreover, we have detailed an overview of some of the many-core platforms that exist nowadays.

We see that many-core systems are here and will continue in the future. Network-on-Chip is the dominant interconnection architecture.

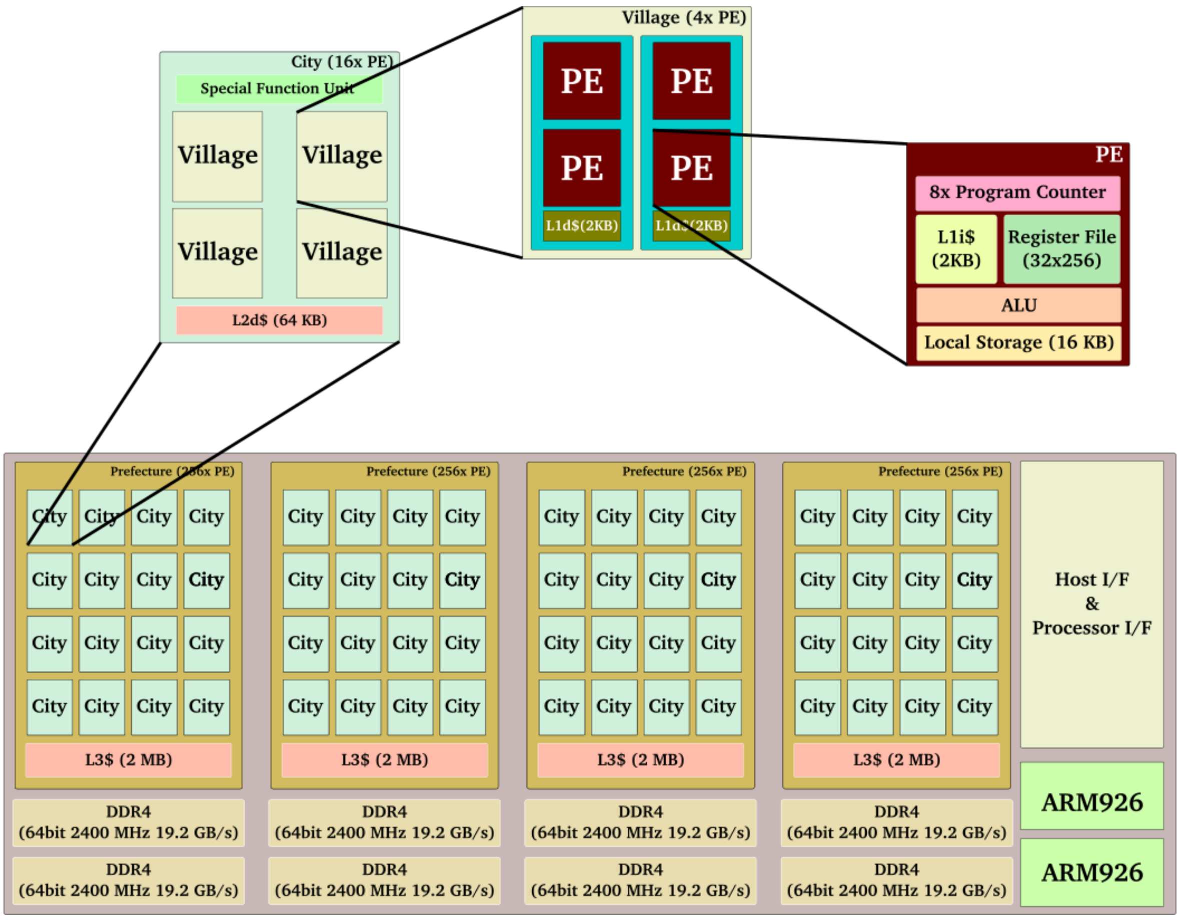


Figure 2.15: PEZY-SC processor architecture

Chapter 3

Cost Model, Technology Node Assumptions, and Methodology

This chapter presents the cost models for the link complexity, routers, buffers, and area space for three topologies: the mesh, the concentrated mesh (cmesh), and the fat quadtree (FQT). In addition, it presents the technology node assumptions we made using data from the International Technology Roadmap for Semiconductors (ITRS) for the year 2023. Moreover, this chapter compares a number of Network-on-Chip (NoC) simulation tools and explains in detail how the Heterogeneous Network-on-Chip Simulator (HNOCS) works and why we selected it as a simulation tool for this research work. HNOCS-XT is our extended version of HNOCS. We made a lot of modifications and additions to it to suit the requirements of this work, which will be detailed later. This chapter describes the experimental methodology we use in this research to measure the performance of different topologies by enforcing locality.

3.1 Topological Properties

In this section, we present the cost model of the mesh, the concentrated mesh, and the fat quadtree in terms of link complexity, number of routers, and buffers. Table 3.1 shows the notations used for the cost model throughout this section.

Table 3.1: Table of notations

Symbol	Description	Symbol	Description
N	Total number of cores	N_L	Total number of links
n_B	Number of buffers	N_R	Total number of routers
n_{VC}	Number of virtual channels	N_B	Total number of buffers

3.1.1 Link Complexity

Link complexity is the total number of links in the topology. A link is a set of wires that connects two routers in the network. Links may consist of one or more physical channels. Table 3.2 shows that the mesh and the concentrated mesh have fewer links than the fat quadtree. For a fair comparison between the topologies, virtual channels are added in the mesh and the concentrated mesh, while the fat quadtree has no virtual channels.

The wire link overhead is how much space the wires take from the chip area. To calculate the wire link overhead, we get the link's width $Link_{width}$ as in Equation 3.1, where (W) is the wire pitch ($pitch = width + space$ [69]), (N_{bits}) is the number of bits in parallel for one packet, and N_{layers} is the number of layers.

$$Link_{width} = \frac{W \times N_{bits}}{N_{layers}} \tag{3.1}$$

The number of vertical wires in a fat quadtree can be obtained $Vertical_{wire} = 2^{(log_4 N-1)} log_4 N$, and for the mesh and the concentrated mesh $Vertical_{wire} = \sqrt{N}$ and $Vertical_{wire} = \frac{\sqrt{N}}{2}$, respectively, where N is the number of cores. From Equation 3.1 and the number of vertical wires, the total width of the vertical link is:

$$Width_{VerticalLink} = Link_{width} \times Vertical_{wire} \tag{3.2}$$

The width of the chip is

$$Width_{Chip} = Length_{Core} \times \sqrt{N} + Width_{VerticalLink} \tag{3.3}$$

Starting from Equation 3.1, the total width of the vertical link and the width of the chip, we can compute the area overheads as follows:

$$Area_{Wire} = 2 \times Width_{VerticalLink} \times Width_{Chip} \tag{3.4}$$

$$Area_{Chip} = Area_{Core} \times N + Area_{Wire} \tag{3.5}$$

$$AreaOverhead = \frac{Area_{Wire}}{Area_{Chip}} \tag{3.6}$$

Table 3.2: Cost model for N cores

	Mesh	Fat Quadtree	Concentrated Mesh
N_L	$2\sqrt{N}(\sqrt{N}-1).n_{VC}$	$N.log_4(N)$	$\sqrt{N}(\frac{\sqrt{N}}{2}-1).n_{VC}$
N_R	N	$\frac{N-1}{3}$	$\frac{N}{4}$
N_B	$5n_Bn_{VC}N$	$n_B(N-4)(\sqrt{N}-2)+2n_B\sqrt{N}$	$2n_Bn_{VC}N$

3.1.2 Routers

A router connects two or more links using ports. Radix is the number of ports a router has. The router decides which link the packets should go to next in order to reach the destination, based on a specific routing algorithm depending on the network topology. The number of routers in the mesh has the order of $O(N)$ compared to $O(N/4)$ and $O(N/3)$ for the concentrated mesh and the fat quadtree, respectively. The mesh has a radix of five ports while the concentrated mesh and the fat quadtree have eight and five ports, respectively.

3.1.3 Buffers

The total number of buffers in the mesh and the concentrated mesh is straightforward, as in each router there are five and eight ports, respectively. The total number of buffers in the fat quadtree is more complex since the buffer size is not constant at each port for each router. The buffer size doubles at every level because the wire lengths are doubling at every level. The root router has only four ports while the rest of the routers have five ports. Therefore, the mesh and the concentrated mesh have the order of $O(N)$ number of buffers compared to $O(N\sqrt{N})$ in the fat quadtree. Table 3.2 shows the cost model for all three topologies.

3.2 Technology Node Assumptions

In this section, we assume that the technology node used in the research is the $10\,nm$ process as projected for the year 2023 by the International Technology Roadmap for Semiconductors (ITRS). ITRS is a set of documents produced by a group of semiconductor industry experts. The documents represent best opinion on the directions of some research areas of technology such as integrated circuit (IC) interconnects, yield enhancement, and process integration devices and structures. Moreover, the documents include timelines and predictions into the future, up to around 15 years [70]. We use the data from ITRS to ensure realistic simulation results of the next decade's many-core systems. Table 3.3 lists the physical parameters for this technology node from the 2011 ITRS data [70]. Because semiconductor companies rarely publish the real size of a die, we need to estimate it. We use the die size ($433.5\,mm^2$) of the 64-core Tile64 from [71] to estimate the current core size and scale it to the year 2023

Table 3.3: Technology parameters for 10nm CMOS (2023) based on ITRS 2011

Year	2013	2023
Chip size at production	$140\,mm^2$	$111\,mm^2$
Global wire delay	$1\,ns/mm$	$33.8\,ns/mm$
Estimated core size	$6.8\,mm^2$	$0.3\,mm^2$
Estimated wire delay	$2.6\,ns$	$17.5\,ns$

technology node. From ITRS [70] the chip size at production in 2013 was $140\,mm^2$ so it is less than the estimated core size in [71] by a factor of 3.1 $(433.5/140)$. In 2023, the chip will contain $20\times$ more cores than today's chip, as Tile64 has 64 cores. Consequently, the chip size in 2023 with nearly 1280 (20×64) cores will be approximately $3.1 \times 111 = 344.1\,mm^2$. Hence, the area of one core is $344.1/1280 = 0.27\,mm^2$. This core area corresponds to dimensions of about $0.52\,mm \times 0.52\,mm$. The width of one core is thus $0.52\,mm$ and the wire delay can be estimated at $0.52 \times 33.8 = 17.5\,ns$.

3.3 Overheads for a 1024-core chip, 10nm (2023) ITRS Node

We use the cost model mentioned in Section 3.1 to compute realistic overheads for a 1024-core many-core chip. The mesh and the cmesh have $2\,VC$, while the fat quadtree has no virtual channels. The number of buffers is $16\,flits$. The flit's size is $32\,Bytes$. In terms of buffer space overhead, the mesh will require $5.1\,KB$ of storage per core, the cmesh $2\,KB$ and the fat quadtree $15.3\,KB$. Although the total number of buffers (N_B) is a lot more in the fat quadtree than the mesh and the cmesh, it is only a very small fraction of the total size of the chip. For example, the per-core L2 cache on the 60-core Xeon Phi is already $512\,KB$ [72]. With the above assumptions, the area of a $15.3\,KB$ SRAM buffer would be 0.14% of the estimated core size. Since the memory density is $37.6\,MB/mm^2$, then $11.3\,MB$ is the memory size that can be placed on top of each core. In terms of wire overhead, our cost model shows that the wire area overhead for the fat quadtree would be 0.3% of the estimated chip size for a 1024-core chip (wire pitch $17\,nm$).

These results are very important as they indicate that for this type of many-core architecture the NoC overhead is negligible, which means that the choice of the NoC can be based solely on performance.

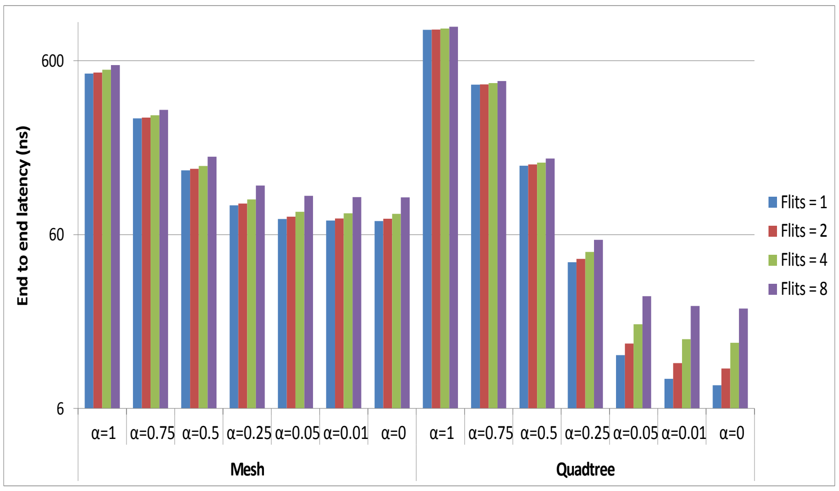


Figure 3.1: Effect of increasing number of flits per packet on the performance ($\alpha = 1$: no locality,$\alpha = 0$: total locality)

3.4 Packet Format and Switching

In this section, we discuss why we use a degenerated case of wormhole switching which uses only a single flit. This has the advantage over wormhole switching with more flits per packet because it will not block the routers along the packet path. Lee et al. [73] argued that widening the flit size will increase the Network-on-Chip performance and make it cost effective. In addition, in a modern chip wires are cheap, see Section 3.3.

In Figure 3.1, we show the effect of using different flits per packet when enforcing locality on the mesh and the fat quadtree. The locality parameter is α. When $\alpha = 0$, it means that all of the packets are sent to neighbouring cores and when $\alpha = 1$, it means that all of the packets are sent to further cores. The figure shows that the average packet latency of the system improves when reducing the number of flits per packet. The fat quadtree with single-flit packets performs 15% to 75% better compared to 4 flits per packet under high locality traffic $(\alpha < 0.25)$. The reason is when more flits are in a packet, more routers will be occupied by one packet which leads to increased latency.

3.5 Simulation Tool

We discuss a number of NoC simulation tools and explain in detail the simulation tool selected for this work. In addition, we list the extensions and modifications made in the selected simulation tool. In general, building a real system for evaluation is too costly and not practical. In addition, it is impossible because we assume a technology node for the year 2023 and it does not exist yet. Hence, building a simulation model is the most suitable approach for evaluating and studying many-core systems. Furthermore, building a simulation tool is cost effective and a feasible way to experiment and analyse theories, algorithms, mechanisms, or strategies of the actual system. An analytical model for hierarchical NoC is difficult and impossible to do; however, some analytical models are used for wire delay and chip area, see Section 3.1.

Simulators are essential tools in evaluating the performance of different NoC designs and new proposals. Table 3.4 presents a comparison between various NoC simulators. NIRGAM is a SystemC based discrete event, cycle accurate simulator for research in Network-on-Chip. It provides substantial support to experiment with NoC design in terms of routing algorithms and applications on various topologies [74]. BookSim is a cycle-accurate interconnection network simulator. It supports a wide range of topologies such as mesh, torus, and flattened butterfly networks. It provides diverse routing algorithms and includes numerous options for customising the networks router microarchitecture [75]. Noxim is an open, extensible, and cycle-accurate Network-on-Chip Simulator developed using SystemC. Noxim has a command line interface for defining several parameters of a NoC in mesh topology [76].

The selected framework for this research is OMNeT++ (Objective Modular Network Testbed in C++), which is an object-oriented modular discrete event simulator. It is a very popular simulator, which has been in development since 1992. Its primary use is to simulate communication networks and other distributed systems. One important feature in OMNeT++ is its generic and flexible architecture. Therefore, it is used in other areas, like simulation of IT systems, queuing networks, hardware architectures, and business processes, as well [77]. OMNeT++ is a public-source, component-based, modular, and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. It also runs on different operating systems, such as Linux, other Unix-like systems, and Windows. This has allowed OMNeT++ to rapidly become a popular simulation platform in the scientific community as well as in industrial settings [77].

OMNeT++ has several packages and technology specific simulators built on top of it, such as HNOCS. HNOCS is an open-source NoC simulator based on OMNeT++. HNOCS stands for Heterogeneous Network-on-Chip Simulator. This simulator was built to provide researchers with an open-source, scalable, extendable, and fully parameterisable framework for modelling NoCs [16]. Section 3.5.1 explains in detail how HNOCS works, and Section 3.5.2 mentions HNOCS-XT, which is an improved HNOCS with extensions and modifications that we made to suit our research.

3.5.1 Heterogeneous Network-on-Chip Simulator (HNOCS)

The HNOCS simulator uses mesh topology and wormhole switching with virtual channels and credit based flow control. The XY routing algorithm is used to route the packets.

Table 3.4: NoC Simulation

	Frame work	Topologies
NIRGAM [74]	SystemC	All
BookSim [75]	C++	All
Noxim [76]	SystemC	Mesh
HNOCS [16]	OmNet++	Mesh
HNOCS-XT	OmNet++	All

HNOCS provides a set of statistical measurements such as the throughput and different latency statistics at the flit and packet levels. HNOCS is selected to simulate Network-on-Chip because it has a modular structure that makes it easier to modify and to add compared to other simulators.

In HNOCS, a NoC is built from two main modules: routers and cores. The core (Fig 3.3) contains the source where the messages are generated and the sink where the messages are received and statistical values are collected. The router is hierarchically built as a collection of connected ports. The capacity and number of Virtual Channels (VCs) can be configured at each port. The router (Fig 3.4) is made of a number of ports. The port (Fig 3.5) consists of the input port (inport), output port selector (opCalc), VC-allocator (vcCalc), and scheduler (sched). In the input port, the incoming flits are stored in the proper VC FIFO buffer. Then, the head-flit is sent to the output-port selector to calculate the output port according to a routing algorithm. After that, the VC-allocator allocates a proper output VC. The scheduler arbitrates between the different packets that need to be transmitted through the proper outport.

HNOCS Simulator Architecture

The simulator architecture below shows how the hierarchical modules in HNOCS are constructed. A complete network topology is built from routers and cores, and it describes the connections and links between them.

- Core (Fig 3.3)
  - Source: packet generator
  - Sink: packet collector
- Router (Fig 3.5)
  - Ports (Fig 3.5)
    - ∗ In-port: FIFO queue per VC

  - · vcCalc: VC-allocator
  - · opCal: routing algorithm (XY routing)

- ∗ Sched: Arbiters for selecting which packet wins the output port using wormhole switching

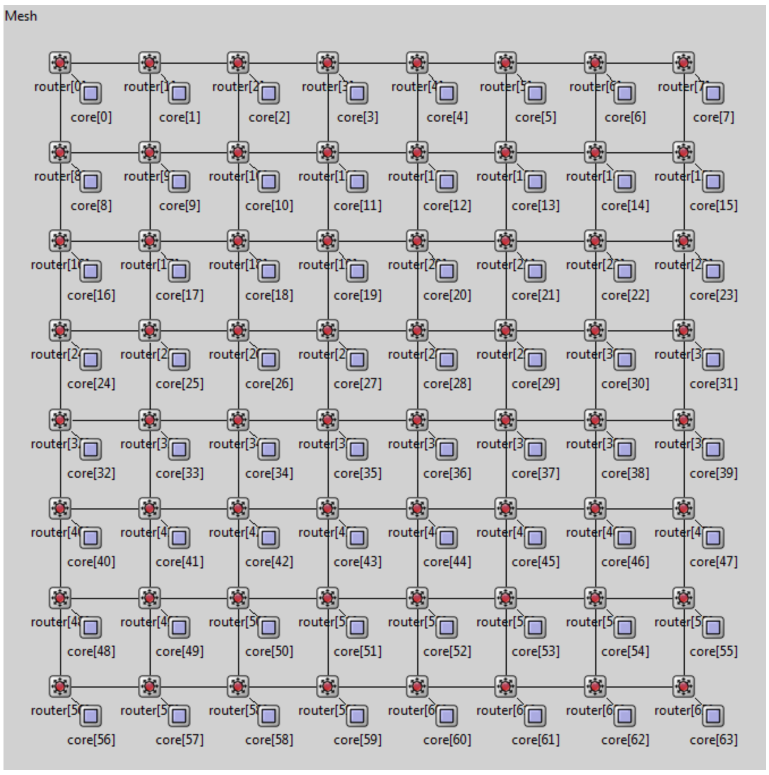


Figure 3.2: Mesh 8x8 as displayed in OmNeT++ when running HNOCS

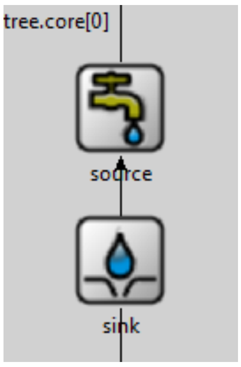


Figure 3.3: The core consists of the source where the messages are generated and the sink where the messages are received and statistical values are collected

Router Pipeline

The router pipeline describes the process of a head-flit when it arrives at the router.

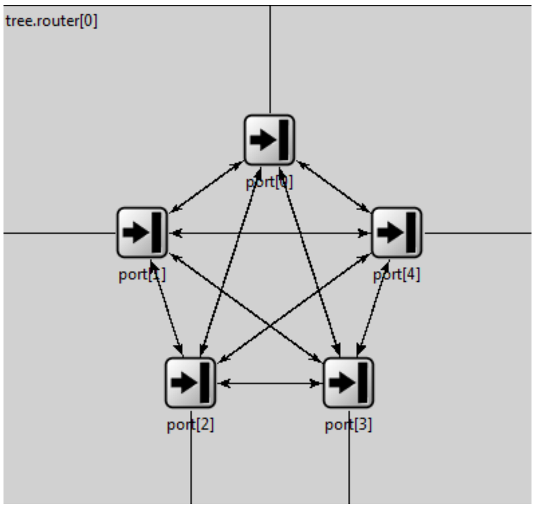


Figure 3.4: The router in a mesh consists of five ports: four ports to the neighbouring router and one port to the core

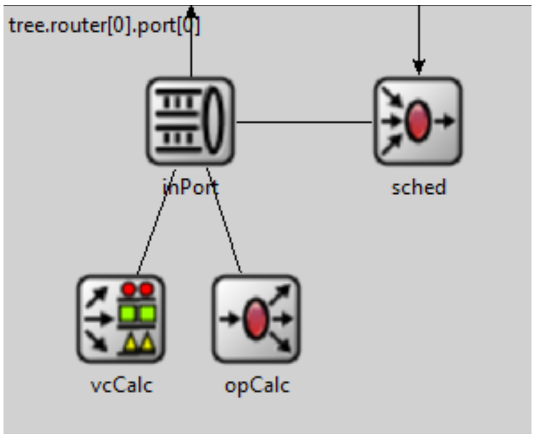


Figure 3.5: The port consists of input port (inport), output port selector (opCalc), VC-allocator (vcCalc), and scheduler (sched)

1. When a head-flit arrives at an in-port, it is queued in the proper VC buffer and the routing module calculates the packet's out-port.
2. When the head-flit is in the head of the buffer, the VC-allocator assigns a VC over the out-port.
3. The in-port sends a request message to the scheduler of the out-port.
4. Once the scheduler can transmit the head-flit, it sends a grant message to the in-port.
5. The in-port sends a credit message to the scheduler of the previous router.
6. The in-port sends the head-flit to the out-port.
7. The scheduler sends it to the in-port of the next-router.
8. The rest of the packet's flits are assigned the same out-port and VC of their head-flit. The scheduler sends grant messages to the in-port whenever it can transmit the packet's flits. In return, the in-port sends the next packet's flit or NACK if there are no ready flits to transmit.

The scheduler serves the request messages from the in-ports (per VC) in a FIFO manner. It employs a round-robin (each VC with outstanding flits sends a single flit) or a winner-takes-all arbitration (the winner VC sends all its outstanding flits). Flits have to pass scheduling: $REQ \rightarrow GNT \rightarrow ACK/NAK$.

3.5.2 Heterogeneous Network-on-Chip Simulator eXTended (HNOCS-XT)

The original HNOCS supports only mesh topology with XY routing. For the purpose of our research we created an extended version of HNOCS called HNOCS-XT. HNOCS-XT supports cmesh and fat quadtree topologies and their routing algorithms.

The hierarchical cache model routing for all three topologies was built since they require a different routing policy than the normal point-to-point routing. Moreover, a cache coherency protocol was implemented with the additional message types it requires, which are eviction and invalidation messages.

HNOCS was built for two or more flits per packet; therefore, some modifications were required in order to accept one flit per packet. Moreover, locality-based traffic models were added as well. No physical parameters of the NoC were built in HNOCS; using ITRS data, the physical parameters of the NoC for the year 2023 were integrated in HNOCS-XT.

3.5.3 HNOCS-XT and Noxim

HNOCS-XT was compared and verified with Noxim to assess the accuracy, the running time, and the correctness of the simulators. Both simulation tools were setup using the same parameters. Figures 3.6 and 3.7 show the results of the comparison between HNOCS-XT and Noxim for an 8×8 mesh network size. They show that HNOCS-XT has lower latencies and slightly higher throughput than Noxim in higher transmission rates. In higher transmission rates, the networks are in saturation. The network is in saturation when more flits are injected to the network which leads to increase in latency then the latency remains constant because the flits will not be generated. When the network is congested, the generation of the flits slows down or stop generating flits as in NoC no dropped flits should occurs. We are interested in the results for when the network is not congested. When the networks are not congested they produce the same results, which is more significant. Noxim has lower granularity and it is cycle accurate, which means the values in higher transmission rate in Noxim might be more accurate and closer to the physical behaviour. Noxim was not chosen to simulate the Network-on-Chip for our work because it is harder to modify and compile in a short time. In addition, because it is cycle accurate the simulation runs will be very slow.

3.6 Experimental Methodology

There are a number of traffic simulation methodologies that can be used in NoC simulators: full-system simulation, trace-based traffic, and synthesis traffic. Full-system simulators

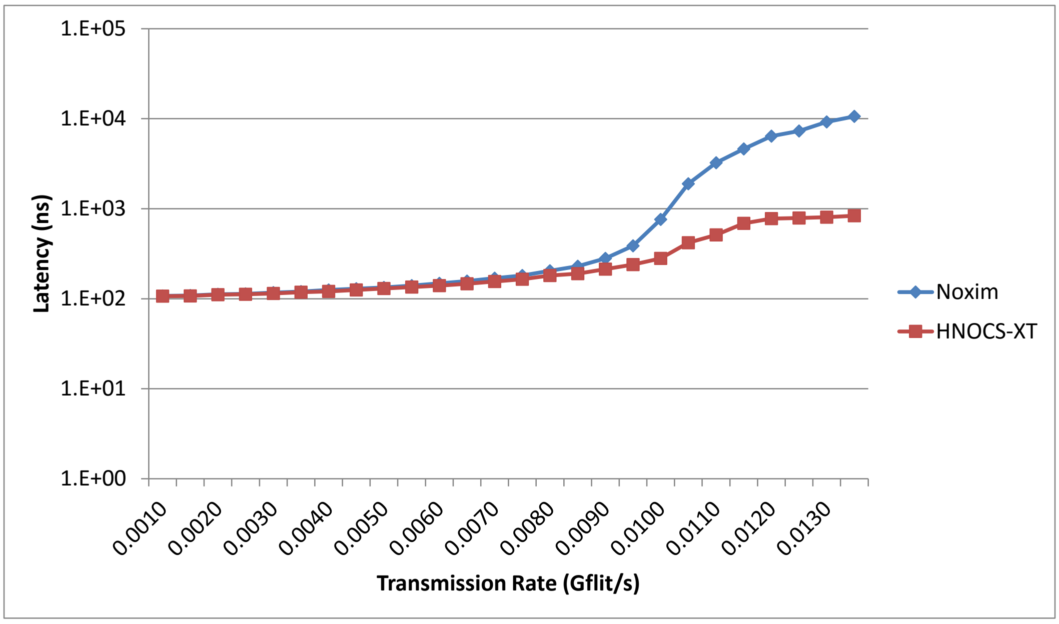


Figure 3.6: Latency comparison between HNOCS-XT and Noxim

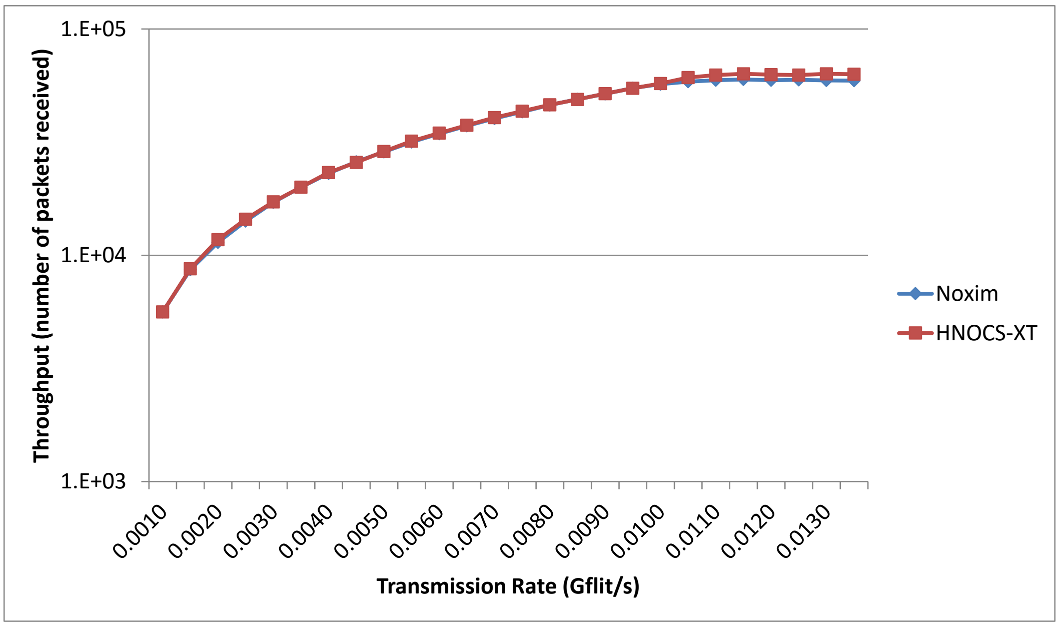


Figure 3.7: Throughput comparison between HNOCS-XT and Noxim

model each hardware component of the overall system and can run full applications and operating systems. As a result, these simulators provide the highest degree of accuracy but have long simulation times. Trace-based traffic using benchmark systems is not scalable for thousands of cores and it is not possible to investigate and research the effect of locality [78]. Traditional synthetic traffic patterns, such as uniform random and transpose, are widely used in NoC research [79]. However, these synthetic traffic patterns do not represent locality enforced architecture.

In order to investigate the effect of the topology on the NoC performance, we propose abstract synthetic traffic generation models that capture the locality behaviour with different distance metrics, which lets us control the degree of locality as well as the shape of the local area and thus provide general insights into the suitability of a given topology for a given locality model.

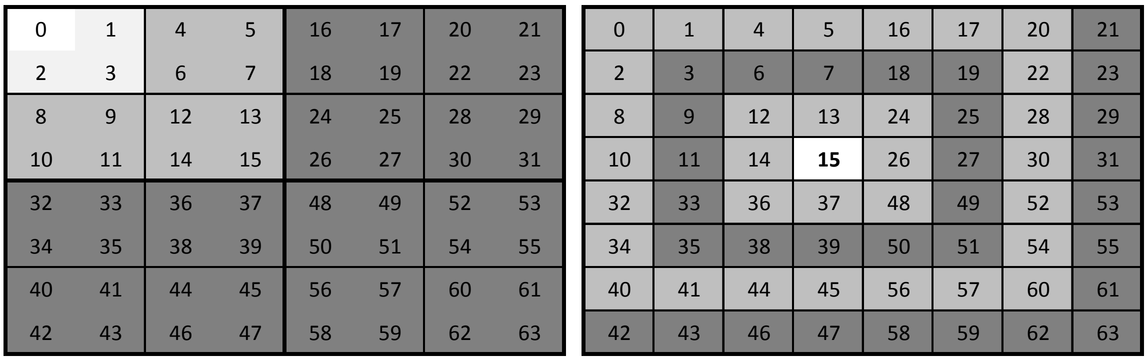
In this section, we propose a simple hierarchical model for locality-based traffic. To model locality, the cores of the chip are grouped and hierarchical groups are created to encompass the whole system. Using $0 < l \leq n$ for the levels of the hierarchy, and assuming $n = log_4(N)$ with N the number of cores, the probability for communication across level-l can be expressed as:

$$\begin{aligned} p(l) &= (1-\alpha)\alpha^{l-1}, 1 \leq l < n \\ p(n) &= \alpha^{n-1} \end{aligned} \tag{3.7}$$

The parameter α relates to the locality of the nodes in the topology. It expresses the probability that a message has to travel a certain distance in the network. Larger α means lower locality: if $\alpha = 0.8$, then according to equation (3.7) 80% of the message will have a destination outside the first cluster, and 80% of that portion outside the second cluster, etc. When $\alpha = 1$, it means that all of the traffic will be sent to the last level cluster. This is worse than uniform traffic where all destinations have an equal probability of being the destination. This type of locality-based model is used to enforce locality and investigate its effect on NoC performance for different topologies.

It is interesting to note that these locality-based models are actually instances of the hotspot traffic model. Hotspot traffic is when messages are destined to a specific core with a certain probability and are otherwise uniformly distributed. According to the hotspot traffic model, each core first generates a random number. If it is less than a predefined threshold, the message is sent to the hotspot core. Otherwise, it is sent to other nodes in the network with a uniform distribution. Our locality-based models perfectly fit this definition.

In this research, we use two different instances of this locality-based model to evaluate the performance of the NoC topologies, group clustering and ring clustering.



a) Group clustering b) Ring clustering

Figure 3.8: Example of the models in an 8×8 mesh

3.6.1 Group Clustering

Group clustering is grouping the cores of the chip per four and creating hierarchical groups to encompass the whole system, in a scale-invariant fashion. In this case $n = log_4(N)$ with N the number of cores, and each level contains 4^l cores. This is a generalisation of the node communication resulting from e.g. tree-based reduction of values from all the nodes. The group clustering locality captures the physical hierarchy of a quadtree topology. Figure 3.8.a shows an example of group clustering of threads where the requesting thread is on core 0.

3.6.2 Ring Clustering

Ring clustering is grouping the cores of the chip in concentric rings around the sender core. In this case, the number of cores per level is $8l$ as long as the rings do not meet the edge. Figure 3.8.b shows an example of ring clustering of nodes where the requested node is core 15. This model is a generalisation of nearest-neighbour stencils typical for many finite-difference based applications such as those encountered in e.g. weather prediction or computational fluid dynamics.

3.7 Performance Measures

We use the performance measures to compare between the different topologies and architectures of the many-core systems in our work. The performance measures we use are latency, and throughput.

3.7.1 Latency

Latency is the time that a packet takes to be transmitted from the source to the destination. Latency in NoC is measured in nanoseconds (ns). Latency is important for measuring the success of NoC is an on-chip network, long latency can significantly worsen the overall application performance, although it has high throughput.

3.7.2 Throughput

Throughput is one of the important parameters that we are interested in measuring in this work. Throughput is defined as the average rate of error-free delivery of data over a communication channel. It is generally measured in bits per second (bps) or data packets per second.

3.8 Summary

In this chapter, we have discussed our assumptions on the technology node used to build the many-core system, the cost model, and our methodology. Moreover, we have detailed the parameters used to simulate a realistic technology node NoC for 2023 from ITRS. We have shown that the buffer and wire space overhead of the NoC on a thousand-core system is negligible for all three topologies, so that the choice of topology depends solely on the performance.

HNOCS was chosen as the simulation tool and it has been extended and modified to HNOCS-XT to suit the requirement of the researched NoC system. HNOCS-XT was explained in detail and it was compared with Noxim for verification and to check the accuracy and the running time of the simulation.

This chapter also introduced the concept of group clustering and ring clustering, which are the two experimental models used to exploit locality in NoC.

Many-core memory architectures can be distributed memory architectures, shared memory architectures, or both distributed shared memory architectures. Many-core distributed memory architectures are when cores communicate using point-to-point messages, such as NUMA architectures. We focus on this type of architecture in Chapter 4. On the other hand, many-core shared memory architectures are when cores share memory and communicate using process-to-memory communication. We focus on this type of architecture in Chapter 5. Many-core shared memory architectures require cache coherency to ensure the consistency of data in their memory; therefore, in Chapter 6 we focus on the effect of cache coherency on many-core shared memory architectures.

Chapter 4

Effects of Locality on Message Passing Communication Performance

This chapter presents the effects of locality on the communication performance in many-core Message Passing Model (MPM). The communication distributed memory architecture NoC is point-to-point. Communication locality can optimise many-core performance since applications mapped on a large many-core system can benefit from clustered communication, where data is placed in cores that are closer to each other. Thus, in this chapter, we show the importance of locality in future many-core systems and that our locality model supports Rent's rule. In addition, we enforce locality using our experimental methodology mentioned in Section 3.6 on flat and hierarchical network topologies to take advantage of such communication locality. Moreover, this chapter describes the topologies used in the simulations, and we compare them in terms of latency and throughput. Furthermore, it illustrates the simulation setup, the results, and the discussion of the many-core NoC simulation results. We show that communication locality improves many-core NoC performance, and the increase in NoC performance depends on the network topology.

4.1 Locality of Computation

Future many-core platforms can be expected to have distributed shared memory architecture [80, 81]. Distributed shared memory architecture uses a hybrid message passing model and shared memory model. Therefore, the communication protocols for programming parallel computers similar to the approach used in NUMA architectures [45] and HPC clusters [82] are to be expected [80]. In fact, it is likely that users will want to deploy legacy MPI code on these novel platforms because rewriting large HPC codebases is a very large effort. However, other message passing approaches, such as Erlang, would be equally suitable for this type of architecture. Regardless of the programming language, it is clear that there is a need for programming models that exploit locality to avoid the long latency of communication between remote cores. We argue that communication cost, not the computation costs, is the dominant factor in processor performance. As we move to thousands of cores on a chip, the physical locality of computation and data becomes critical to performance and cost.

Recently, locality-based traffic and enforcing locality in NoC have been receiving increased attention [83–86]. The performance of future many-core systems with thousands of cores will heavily depend on the Network-on-Chip architecture to provide scalable communication. Communication locality is essential to reduce latency and increase performance. As the number of cores increases, locality will only become more important [83]. Many-core systems should be designed so that cores communicate mainly to the neighbouring cores, to minimise the communication cost.

Without locality, the amount of communication grows with the square of the number of cores. As a consequence, applications running on high-performance compute clusters always display strong locality. Moreover, it has long been known that for high performance, message passing applications require locality. We contend that a combination of locality-aware task placement and a locality-based communication topology can greatly improve performance of message passing style applications.

We define locality as the percentage of flits injected by a core communicating to its immediate neighbours in the network. Applications with high locality tend to have nearest neighbour communication pattern with high local traffic.

4.2 Relationship to Rent's Rule

There is a very interesting relationship between the α in our locality-based model (3.7) and the measure for locality known as Rent's rule, as extended to NoCs by Greenfield et al. in [87]. Referring to Section 3.6, $(1 - \alpha)$ is the locality rate. Group clustering and ring clustering are the models we use to choose the neighbouring cores communicating with each other to enforce locality.

Rent's rule was described in 1971 by Landman and Russo as the relation between the number of terminals at the boundaries of an electronic circuit and the number of internal components such as logic gates [88].

Greenfield et al. [87] argued that network traffic follows Rent's rule. In addition, they argued that Rent's rule will naturally arise in multi-core and many-core because, just as it is typically undesirable to place two blocks on opposite ends of the chip and wire them up, it is also undesirable to map two communicating tasks to tiles at opposite ends. They extended the concept of connection locality in circuits to communication locality among cores, proposing a bandwidth based version of Rent's rule,

$$B = kG^{\beta}$$

where B is the bandwidth sent or received by a cluster of G network nodes, k is the average bandwidth per node, and $0 \leq \beta \leq 1$ is the Rent's exponent.

Heirman et al. [89] showed experimentally that many parallel applications follow Rent's rule. They analysed 13 popular benchmark applications running on 32 and 64 core networks. Using a hierarchically partitioning algorithm they showed that the programs follow Rent's rule with measured values of the Rent's exponent β ranging from 0.55 to 0.74, which proves that communication is definitely localised.

As our many-core architecture is NoC-based, all traffic is transferred over the NoC. We are then principally concerned with the latency and the throughput of our message passing communications and, hence, with the latency and bandwidth of our NoC. In our proposed architecture, the traffic flowing from level l to level $l + 1$ depends on α and the number of cores in the level, 4^l. Thus, for each level, the amount of core traffic generated is proportional to the $B(l) = 4^l(1-\alpha)\alpha^{l-1}$. Following the derivation in [87], we consider the ratio of traffic between two subsequent levels:

$$B(l+1)/B(l) = 4\alpha$$

and generalised to k levels:

$$B(l+k)/B(l) = (4\alpha)^k$$

Following again [87], we define $\alpha = 4^{\beta-1}$ and $x = 4^k$ and set $l = 1$, and we obtain exactly the same equation as in [87], expressing Rent's rule for bandwidth.

$$B(x) = B(1)x^{\beta} \tag{4.1}$$

Thus, we obtain the very interesting result that the traffic bandwidth following from the distribution in (3.7) is governed by Rent's rule. In other words, our proposed hierarchical locality approach expresses Rent's rule for the bandwidth of the generated traffic.

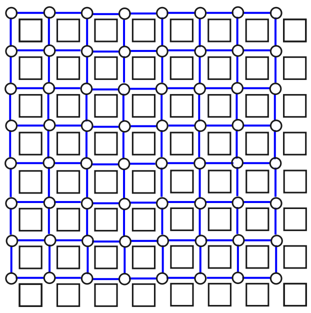


Figure 4.1: Mesh

4.3 Network Topologies

The network topology describes how routers are connected with each other and with the cores. For many-core systems, the communication cost is increasingly important. The topology has a major impact on the scalability and the performance of the network. With this motivation in mind, the network performance and communication locality are analysed for flat and hierarchical topologies. The three different types of network topologies used are the mesh, the concentrated mesh (cmesh), and the fat quadtree (FQT). We choose to use these topologies because chips now are produced using the mesh and the cmesh topologies such as Tilera [64] and PEZY-SC [67]. Xeon-Phi [62] uses ring topology, but this topology will not scale for thousands of cores because the communication latency becomes high, which reduces the overall performance of the many-core system. In regards to the fat quadtree, we will show that under localised traffic it might perform better than the mesh.

4.3.1 Mesh Topology

Figure 4.1 shows the layout for the mesh for an 8×8 network size. The mesh uses XY routing. The wire delay for each link is $17.5\,ns$, as calculated using ITRS data in Section 3.2.

4.3.2 Concentrated Mesh Topology

Figure 4.2 shows the layout for the concentrated mesh for an 8×8 network size. Routing in cmesh is using XY routing in the mesh structure and once the flits arrive to the parent router of the core, the router sends the flit to the appropriate core. The length of the links in the cmesh is double the length in the mesh; therefore, the wire delay for each link in the cmesh is $35\,ns$.

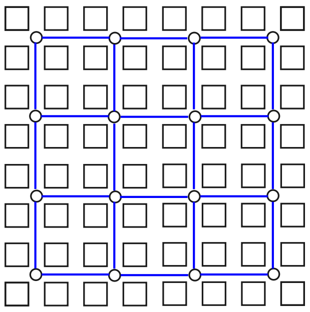


Figure 4.2: Concentrated mesh

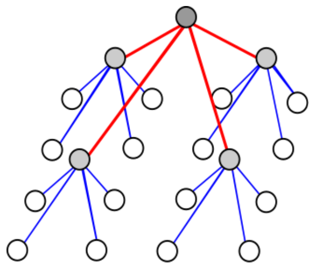


Figure 4.3: Fat quadtree logical layout

4.3.3 Fat Quadtree (FQT) Topology

The fat quadtree uses nearest-common ancestor routing. Figure 4.3 shows the logical layout for a fat quadtree and Figure 4.4 shows the proposed physical layout for a fat quadtree for 64 nodes. In the proposed physical layout, the link length in the fat quadtree doubles as the level of the hierarchy increases. The first level, which is the blue line, has a wire delay of $35\,ns$ and the next level a wire delay of $70\,ns$, and so on.

In Figures 4.5 and 4.6, we calculated the number of hop counts and delays in 1024-core NoC when a packet is generatedfrom all the cores to all the other cores. Frequency refers to the number of occurrences of the hop count or the delay. Although the fat quadtree in a 1024 size NoC has a maximum of 9 hops compared to 63 and 39 hops for the mesh and cmesh, respectively, the fat quadtree has the highest wire delay, as illustrated in Figures 4.5 and 4.6. Because the links between the hops of each level double in length in the fat quadtree,

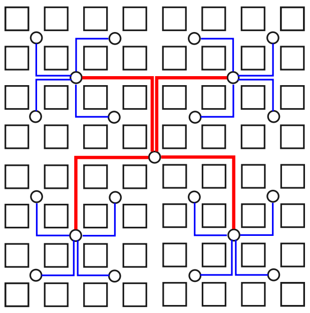


Figure 4.4: Fat quadtree physical layout

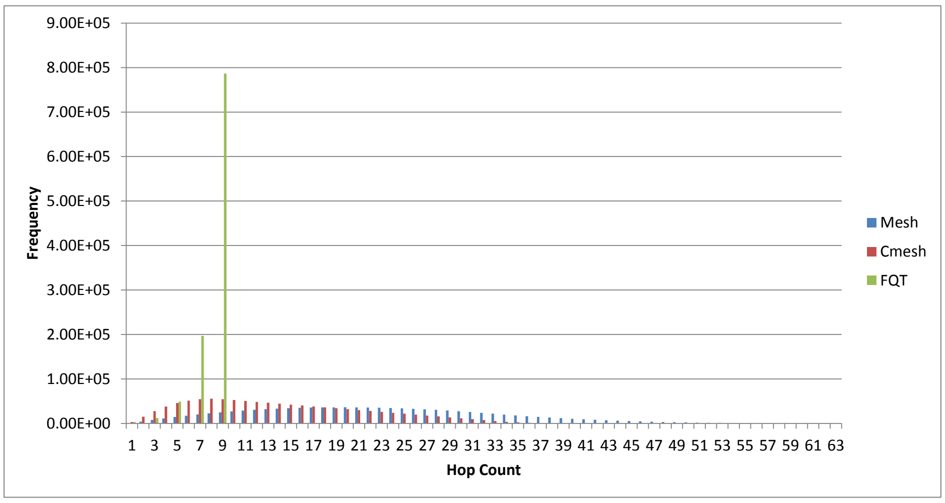


Figure 4.5: Hop count comparison

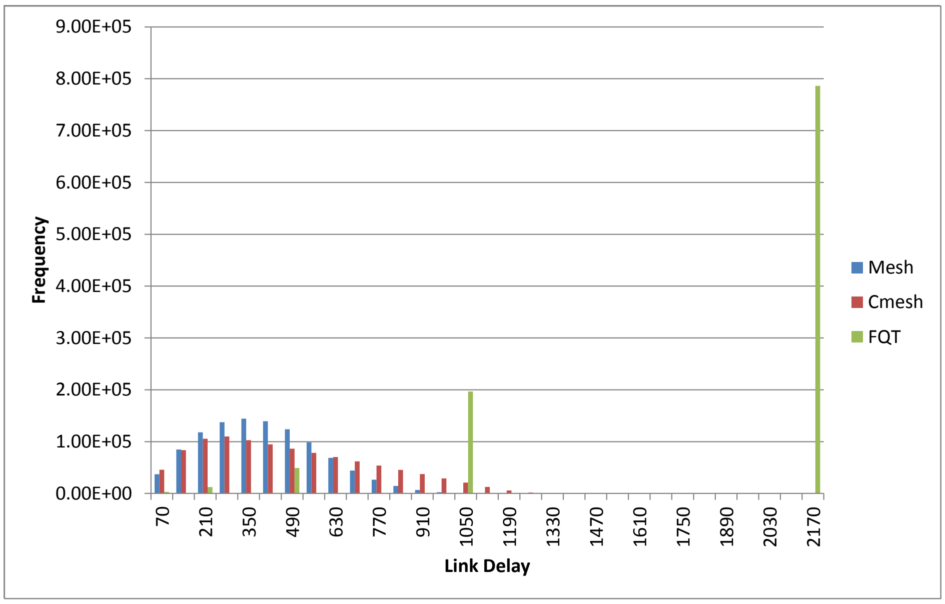


Figure 4.6: Link delay comparison $(17.5\,ns/link)$

the wire delay doubles as well, while the wire delay of the mesh and the cmesh remains constant. Looking at the figures, it is clear that the fat quadtree will have the longest delay, and it will not be scaleable. However, by introducing locality into the picture the fat quadtree can be scaleable and do better in terms of performance.

4.4 Simulation Setup

We simulate the different topologies on a 1024-core chip placed in a regular 32×32 grid using HNOCS-XT. We model the process-to-process communication using Poisson-distributed traffic because it typically offers a good estimate on the average performance of networks and

Table 4.1: Simulation parameters

Topology	Mesh	CMesh	Fat quadtree
Number of virtual channels	2	2	1
Wire delay (ns)	17.5	35	$35 \times 2^{l-1}$, $1 \leq l < n$
Flit size ($bytes$)	32		
Buffer size ($flits/VC$)	16		
Channel datarate (Gb/s)	128		

it has been widely used in the evaluation of interconnection networks. The packet length is one flit and the flit size is $32\,Bytes$. The flit size is equal to the size of most of the current system's messages. The frequency of a modern system is $500\,MHz$; hence, the channel data rate is $128\,Gbps$ and the bus is $256\,bits$ wide. The clock speed is $2\,ns$. The router delay is $2\,ns$. The fat quadtree has more links compared to the mesh and the cmesh. Therefore, two virtual channels are used for the mesh and cmesh while for the fat quadtree one physical channel is used for the lowest-level links, and it quadruples at each level to simulate a fat quadtree. Hence, the fat quadtree has no virtual channels. The buffer size in the router is 16 flits per virtual channel for the mesh and cmesh and 16 flits per physical channel in the fat quadtree. The wire delay is proportional to the distance between the routers, so in a fat quadtree it doubles at each level. The destination is selected using different degrees of localisation. The degree of localisation is described in Section 3.6, which has two different instances of locality-based model: group clustering and ring clustering. Table 4.1 summarises the simulation parameters used in our simulations.

The simulation run time is $1\,ms$. In the simulation, all cores generate traffic at the same time. A core will generate the next flit only if the flit in the source queue was sent; hence, there will be no dropped flits.

4.5 Evaluation and Discussion

We evaluate the performance of the three topologies – the mesh, the concentrated mesh, and the fat quadtree – as a function of the flit generation rate and the localisation degree α in two different locality-based models: group clustering and ring clustering. When $\alpha = 1$, it means there is no locality traffic, as all flits go to the furthest cores to communicate, and when $\alpha = 0$ the traffic is fully local as all the flits go to the neighbouring cores.

4.5.1 Effect of Locality on Latency

For group clustering, Figure 4.7, the results show that the mesh has lower latencies when the degree of localisation is low $(0.5 \leq \alpha \leq 1)$, while the fat quadtree has lower latencies when the degree of localisation is high $(0 \leq \alpha \leq 0.25)$. This indicates that the mesh will perform better when the traffic is uniform and more distributed, but the fat quadtree will perform better when the traffic is localised. It is known that the fat quadtree does not scale well under uniform traffic; however, when locality is introduced the fat quadtree scales. Figure 4.9 clearly shows how the fat quadtree latencies improve when the degree of localisation increases.

The concentrated mesh has low latencies when the traffic is highly localised $(0 \leq \alpha \leq 0.25)$, similar to the latencies in the fat quadtree. This is because the concentrated mesh has four cores for each router, similar to the first level of the fat quadtree. The concentrated mesh has low latencies in low localised traffic $(0.5 \leq \alpha \leq 1)$, similar to the latencies in the mesh. This is because the concentrated mesh has a similar structure to the mesh. However, the concentrated mesh congests faster than the mesh because it has fewer links and a higher wire delay.

For ring clustering, Figure 4.8, the mesh and the concentrated mesh have nearly similar latencies and they perform better than the fat quadtree. The fat quadtree has very high latencies as it congests faster. This is because in most cases the fat quadtree has fewer hops but longer paths with higher delays, and it does not perform well when communicating with neighbouring nodes that do not have the same ancestor.

Overall, the concentrated mesh performs best in group clustering and ring clustering. One might expect the fat quadtree to perform better as the group clustering matches its topology. However, the layout of the fat quadtree results in fewer hops but longer paths, and in the $10\,nm$ CMOS process for the 2023 node the wire delay is dominant ($6.7\times$ worse than for the 2013 node). Figure 4.9 illustrates the latencies of the topologies when the network is not congested and packets are generated every $70\,ns$ for different α parameter. It shows group clustering results have lower latencies than ring clustering; this is an important result for the placement of neighbours in stencil computations.

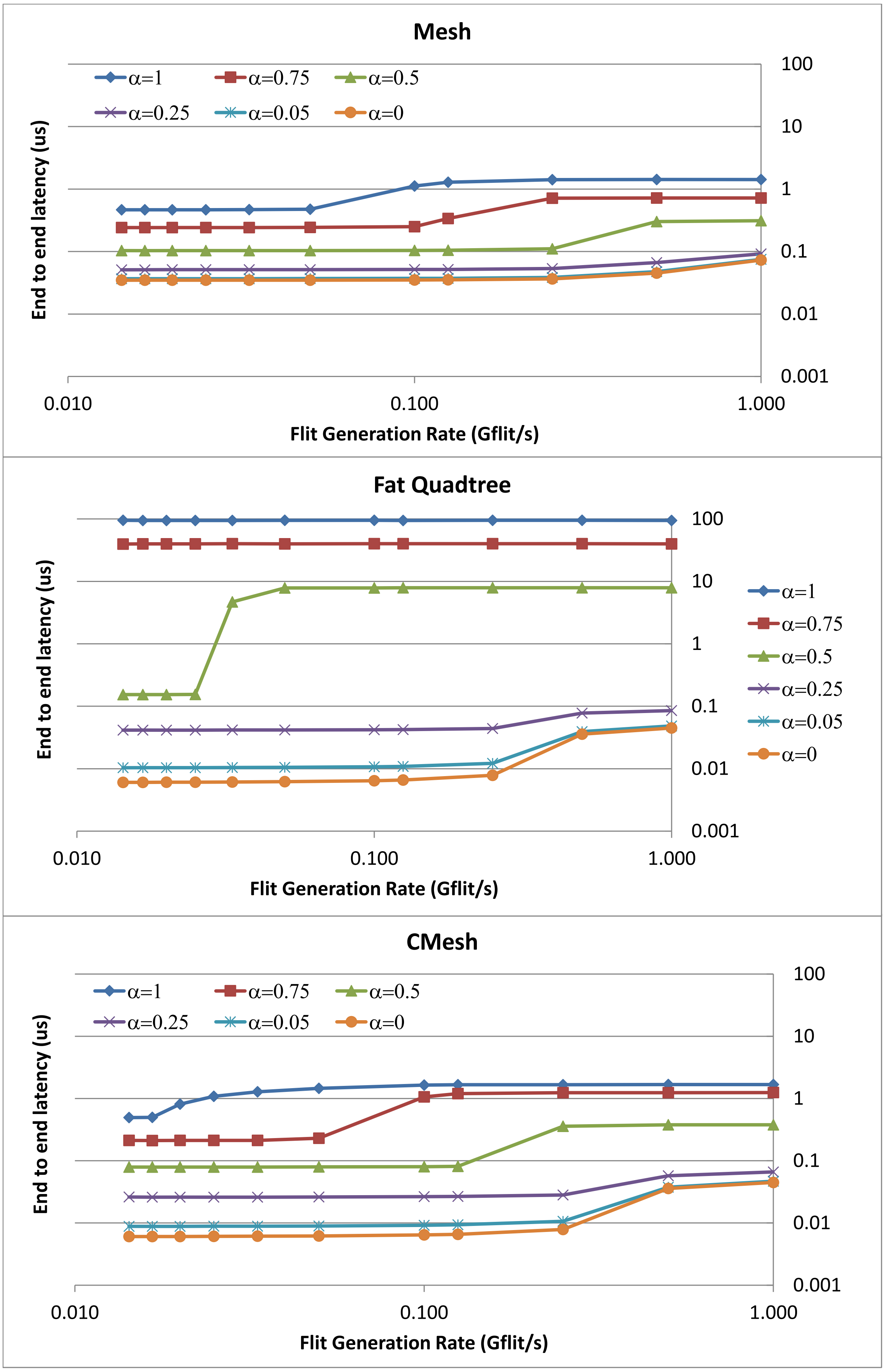


Figure 4.7: Latency for group clustering ($\alpha = 1$: no locality, $\alpha = 0$: total locality)

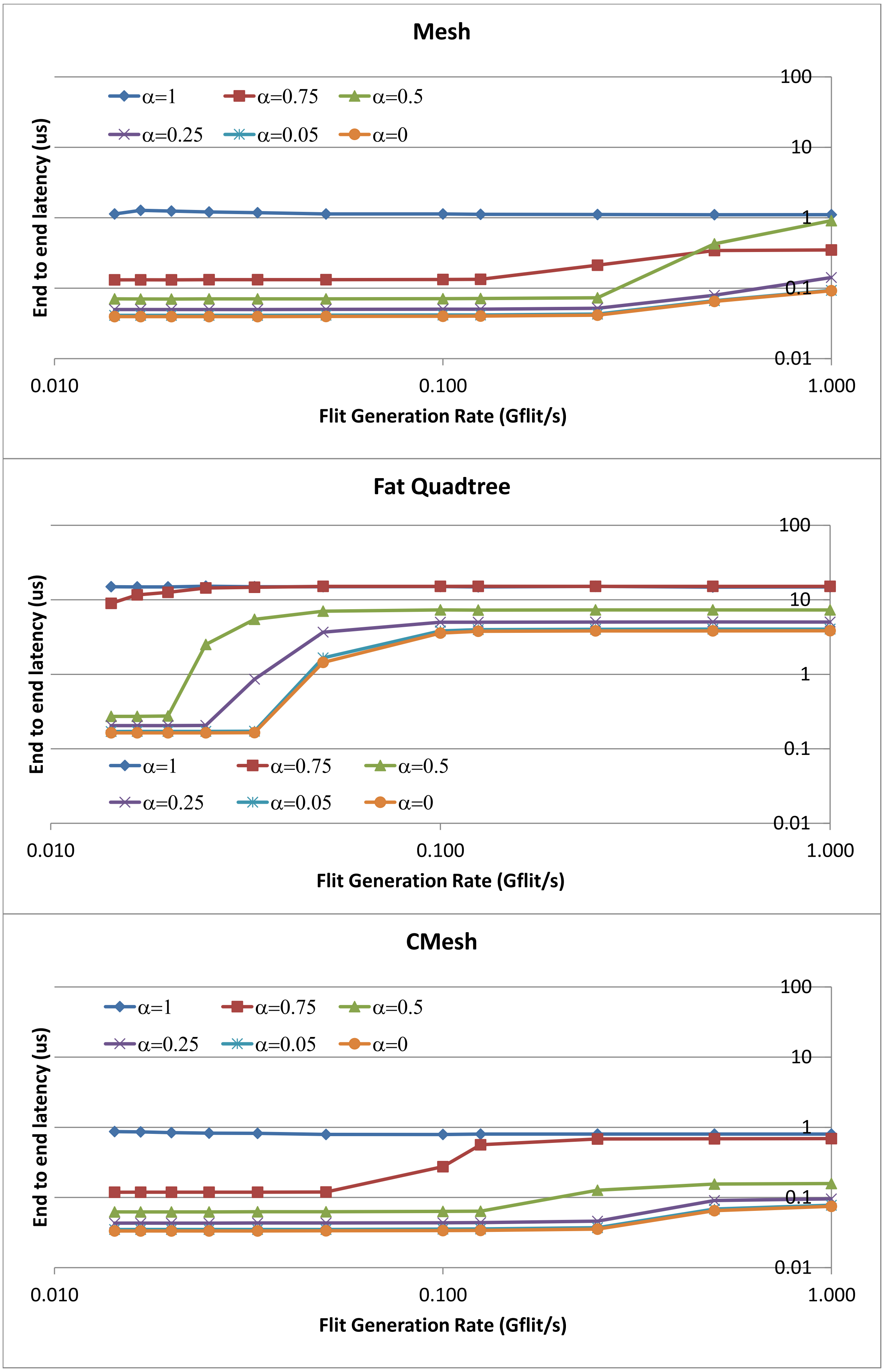
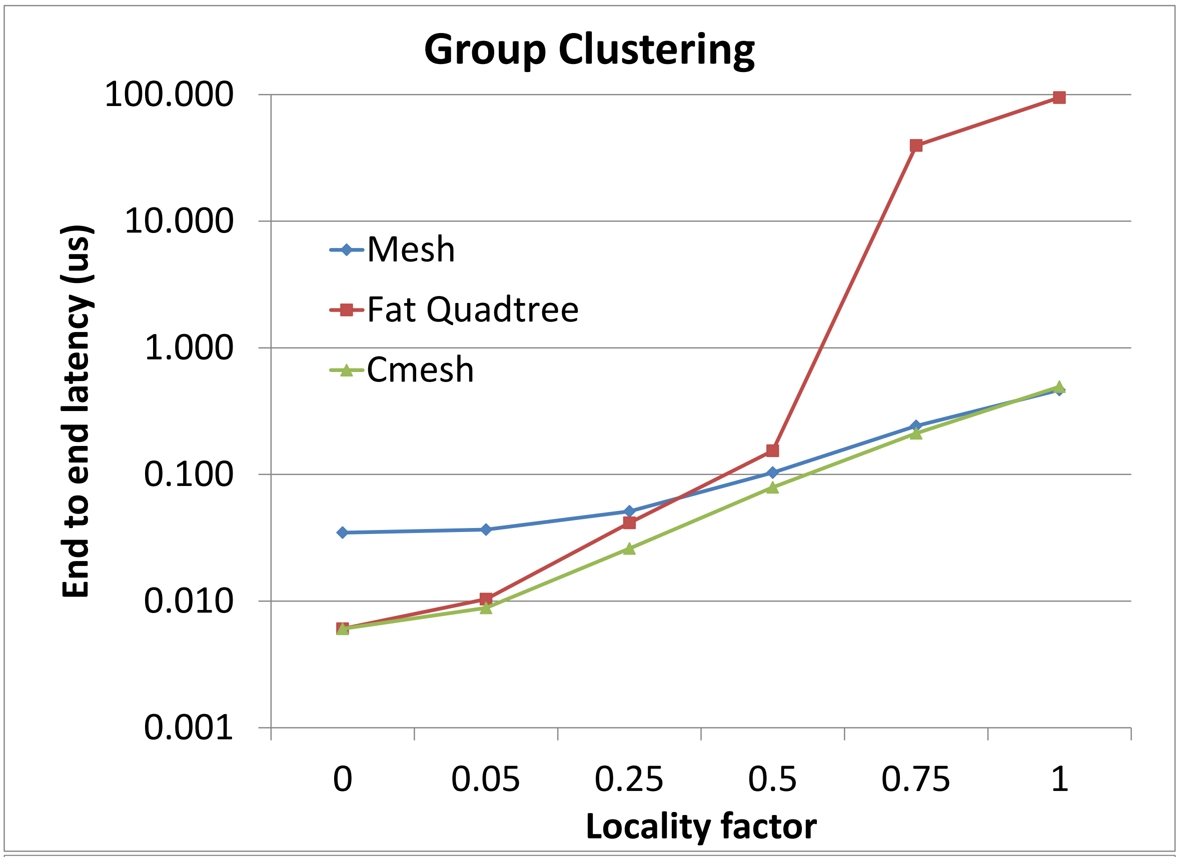


Figure 4.8: Latency for ring clustering ($\alpha = 1$: no locality, $\alpha = 0$: total locality)



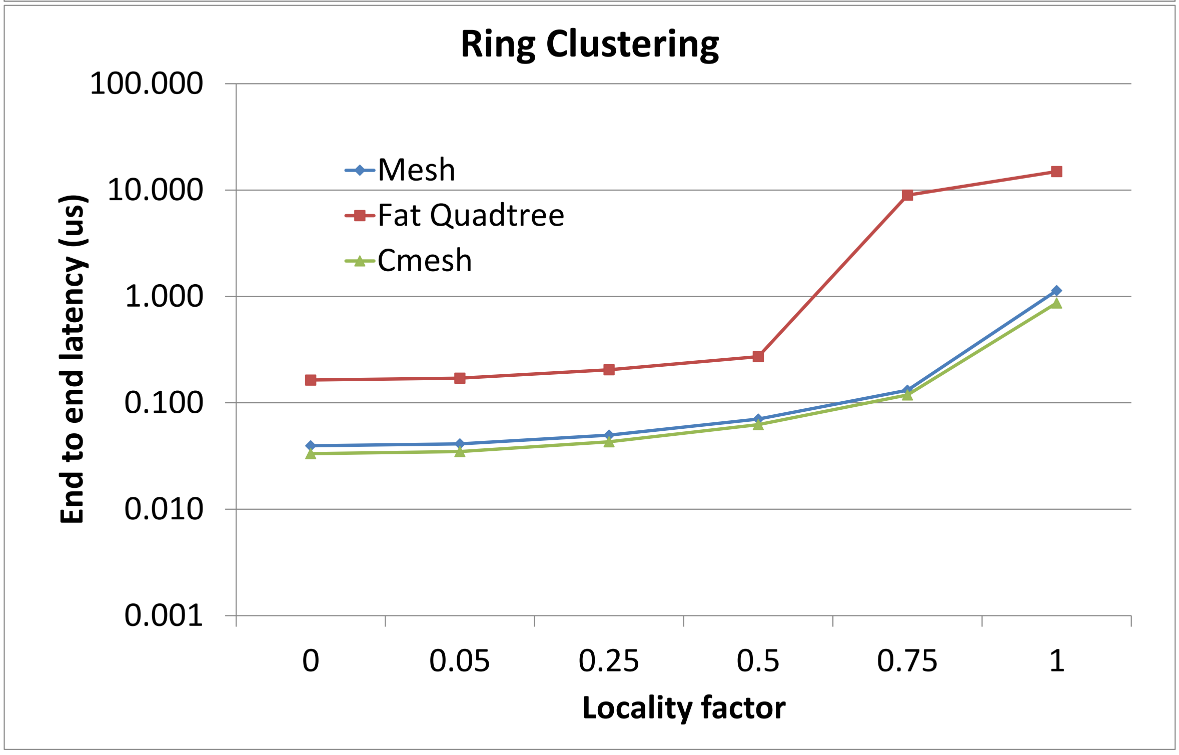


Figure 4.9: Group and ring clustering ($\alpha = 1$: no locality, $\alpha = 0$: total locality)

4.5.2 Effect of Locality on Throughput

For group clustering, Figure 4.10, we observe nearly similar behaviour in all the cases: the throughput increases as the flit generation rate increases. In the case of low locality $(0.5 \leq \alpha \leq 1)$, the throughput increases to a certain point then stays constant because the latencies are higher, and so they are congested and deliver fewer messages. For high locality $(0 \leq \alpha \leq 0.25)$, the throughput is approximately identical in all topologies. For ring clustering, Figure 4.11, the fat quadtree has a lower throughput compared to the mesh and the concentrated mesh, which have nearly similar throughputs.

It is interesting to compare the throughput results of group clustering and ring clustering with today's systems, when our system generates $0.1\,Gflits/s$. In today's systems, PEZY-SC many-core processor has a maximum bandwidth of $1.5\,TB/s$ and Intel Xeon Phi has a maximum bandwidth of $0.5\,TB/s$. Under high locality $(0 \leq \alpha \leq 0.25)$, in all the cases our system has a bandwidth of nearly $3\,TB/s$ except in the fat quadtree for ring clustering, which has a bandwidth of nearly $2\,TB/s$. It is important to mention that the wire delay in our chip in 2023 is $6.7\times$ higher than the wire delay in chips today, as mentioned in Section 3.2.

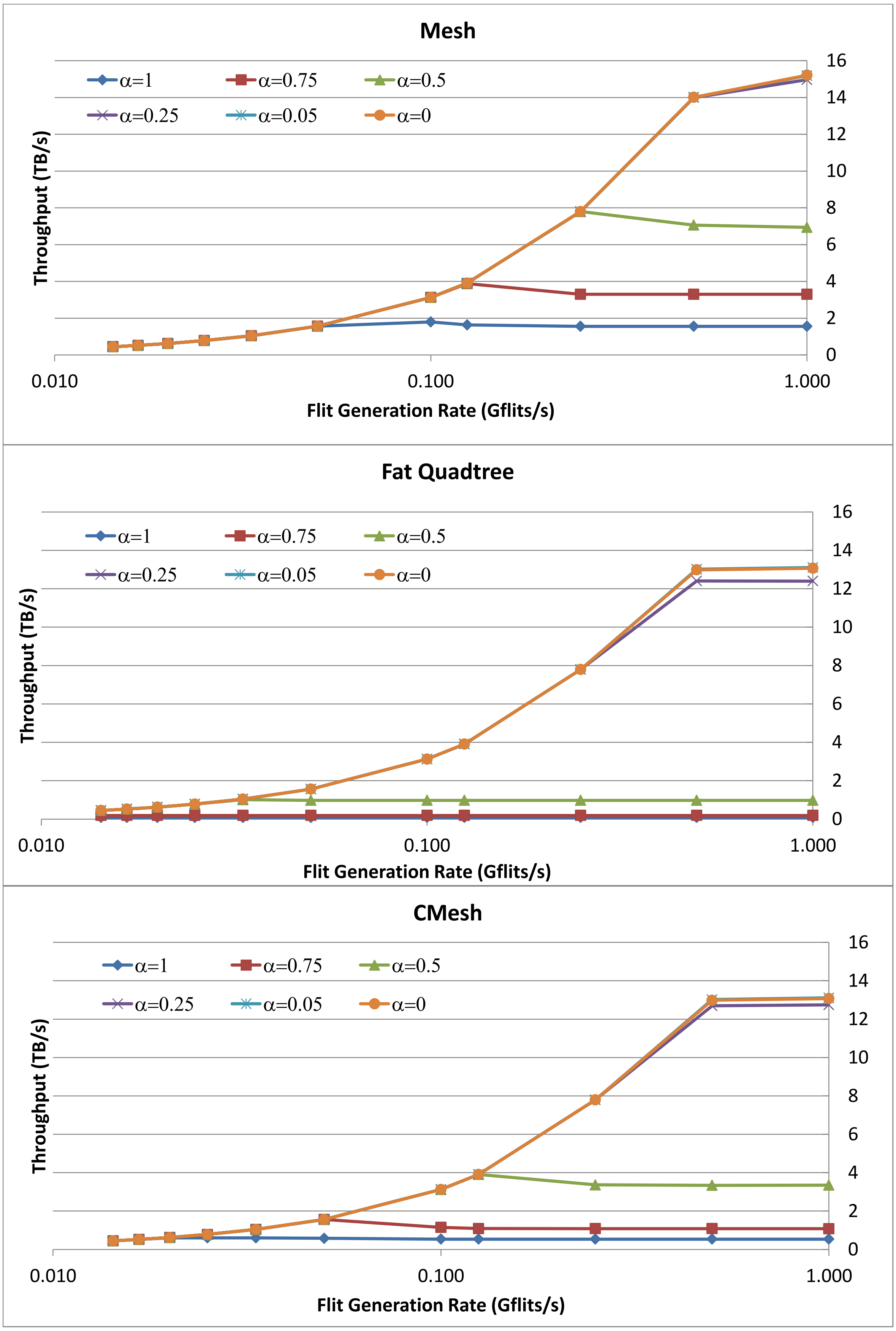


Figure 4.10: Throughput for group clustering ($\alpha = 1$: no locality, $\alpha = 0$: total locality)

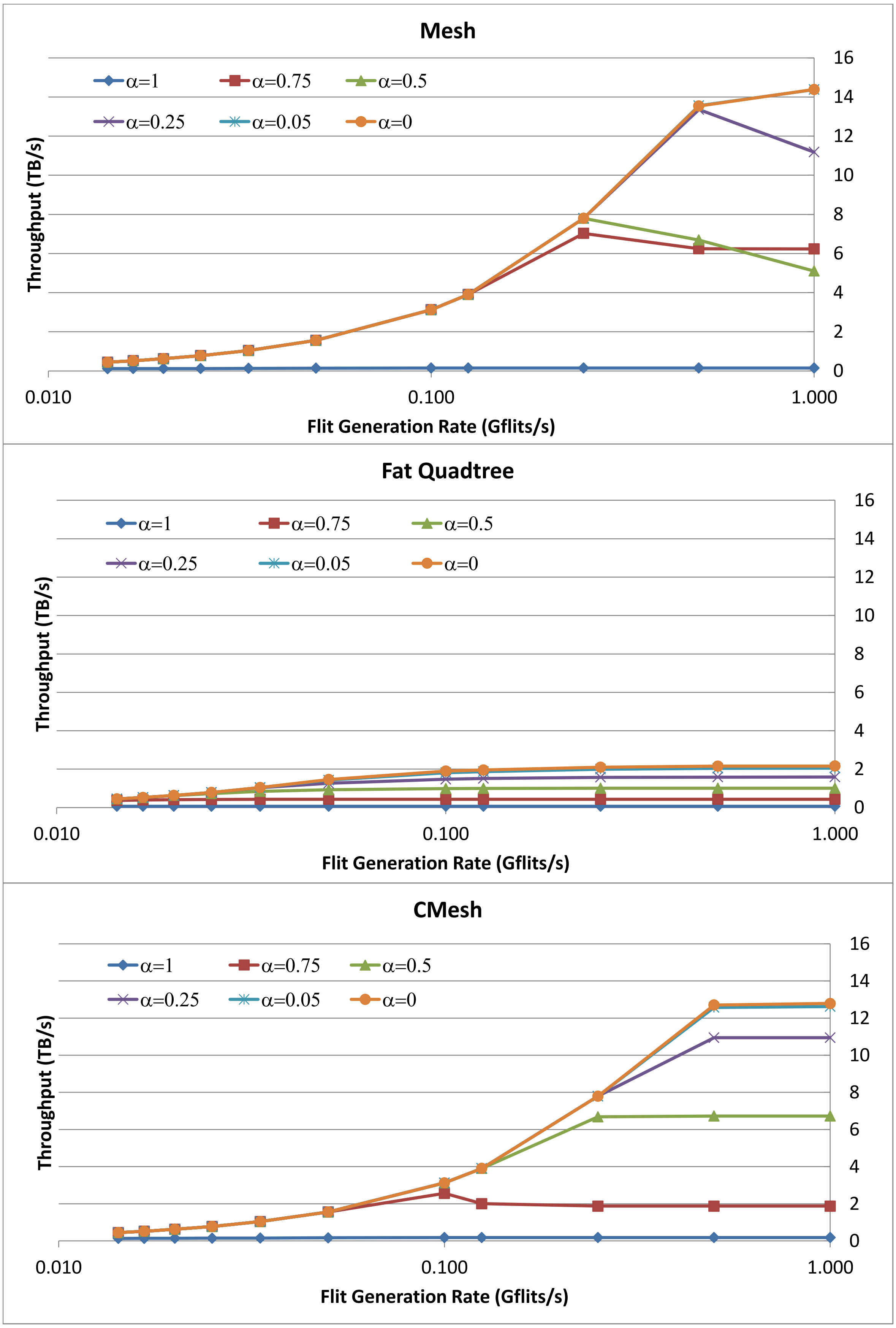


Figure 4.11: Throughput for ring clustering ($\alpha = 1$: no locality, $\alpha = 0$: total locality)

4.5.3 Effect of Locality on Different Network Sizes

Figure 4.12 compares the average latency for different network sizes of 4×4, 8×8, 16×16, and 32×32 for different degrees of localisation. The flit inter-arrival time is $70\,ns$. We observe that for higher locality, the effect of the network size on the latency decreases for both topologies. The fat quadtree does not scale well under no-locality and low-locality traffic; however, it significantly reduces the latency as locality increases.

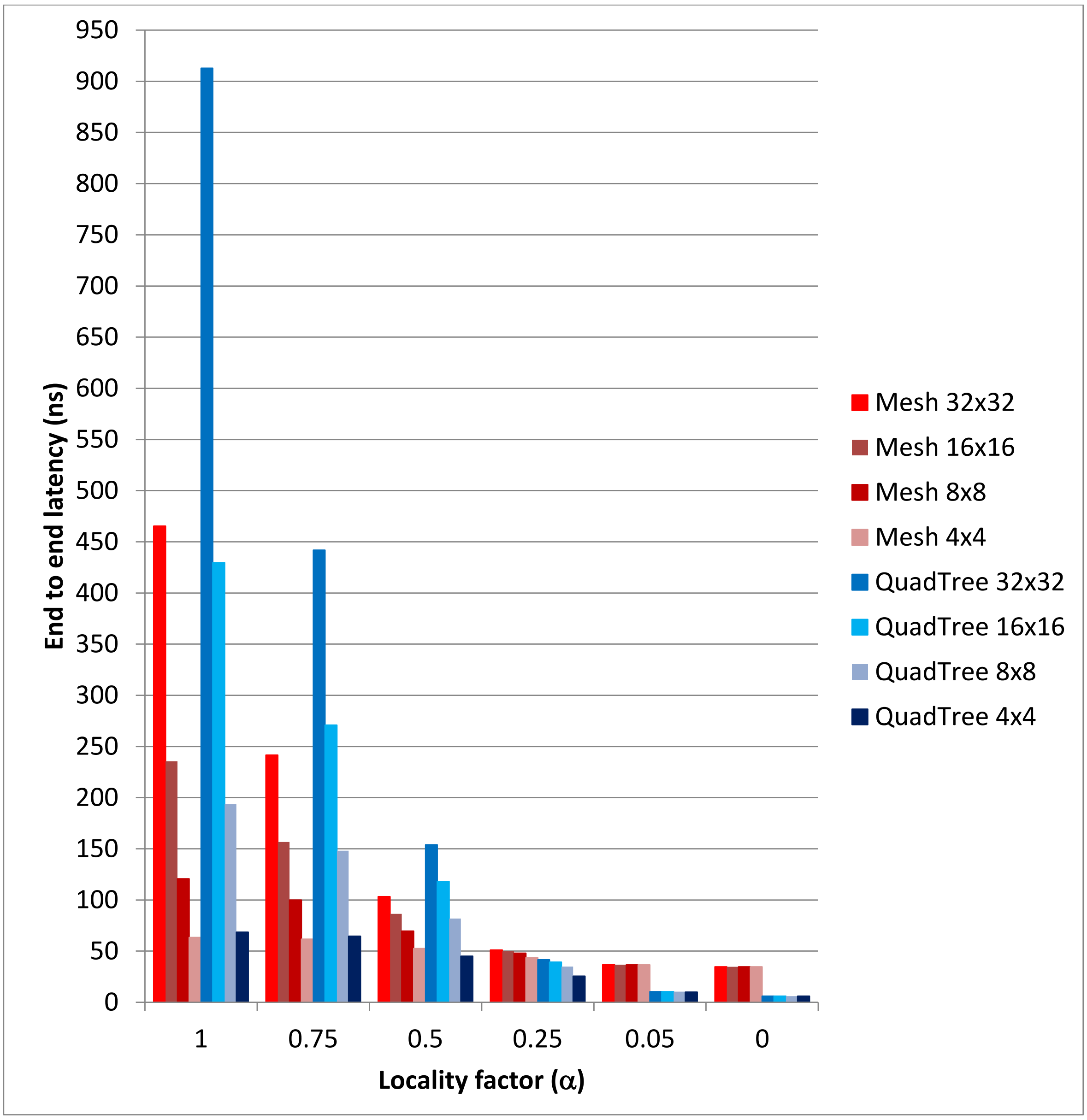


Figure 4.12: Locality versus network size (1: no locality, 0: total locality)

4.6 Summary

In this chapter, we have investigated the overhead and performance of flat (mesh, cmesh) and scale-invariant (fat quadtree) NoC topologies for future many-core systems with thousands of cores under group clustering and ring clustering localisation models for point-to-point traffic. We have shown that the degree of locality and the clustering model strongly affect the performance of the network. Scale-invariant topologies such as the fat quadtree perform worse than flat ones (esp. cmesh) because the reduced hop count is outweighed by the longer path delays, as a consequence of the high wire delay in the $10\,nm$ CMOS process. Our results clearly show the importance of traffic localisation for very large many-core systems. We will next show how these many-core topologies perform in shared memory architecture. Will the flat topology perform better than the scale-invariant topology?

Chapter 5

Effects of Locality on Shared Memory Communication Performance

This chapter presents the effects of locality on the communication performance of shared memory model in many-core systems. Data locality is important in optimising many-core performance as applications mapped on a large many-core system can take advantage of short distant communication, where data is placed in cache banks closer to the cores accessing it. Thus, in this chapter, we design a hierarchical cache model that benefits from communication locality. Moreover, this chapter examines the hierarchical placement of caches on the proposed hierarchical cache model into different topologies. In addition, it illustrates the parameters used for the simulations and compares the performance of the many-core architecture in terms of latency and bandwidth using the ITRS physical data for 2023. The results show that the model of thread placement and the distance of placing them significantly affect the NoC performance. Furthermore, they confirm that scale-invariant topologies perform better than flat topologies.

5.1 Shared Memory Communication

In a NoC-based many-core system with global shared memory, the communication is between the cores and the memory. It means that with shared memory processing cores can communicate with each other by storing data in shared memory locations and subsequently reading and writing into them. The actual communication takes place over the interconnection network.

To make effective use of the resources provided by many-core processors, parallel programming is essential. As a result, applications have increasing numbers of threads, and thread placement and inter-thread communication have become important topics of research. Besides, as the number of cores increases, memory locality becomes even more important. To minimise the communication cost, these many-core processors should be designed so that data accessed by threads are allocated to cache banks or memories closer to the processing core on which a thread is running.

Shared memory traffic and point-to-point traffic are two different traffic patterns [90], and the resulting NoC performance is very different. Chapter 4 discussed the point-to-point traffic while this chapter deals with shared memory traffic. The main focus in this chapter is to examine the effect of shared memory traffic and thread placement on the hierarchical cache architecture, as well as the network performance resulting from different topologies and locality models.

In the presence of caches, the traffic is determined by both the location of the cache with respect to the thread and by the distribution of data over the cache hierarchy. For example, in a distributed cache system with hashing such as that used by the Tilera TilePro [2], the average distance from each core to the cache is constant. In the MIC architecture [3], the L3 cache is the union of the L2 caches for each core. The average distance of a ring topology in a NoC means the memory distance between two threads that are located on physically adjacent cores, which can be up to half the total number of cores. Neither of these architectures can scale to thousands of cores because the average distance grows as the number of cores grow. In addition, the average access time to memory grows as well, which leads to having a slow system.

Memory locality is an essential concept of caching. In order to investigate the effect of memory locality on performance, we propose abstract models of placing threads of shared memory applications with different distance metrics in a hierarchical cache architecture. These models let us control the distance between the threads that share a memory location as well as the shape of the local area, and thus provide general insights into the suitability of a given topology for a given shared memory thread placement model.

Our assumption is that the future many-core architecture will have a distributed shared memory architecture, with memory embedded in the processor tiles or stacked on top of them (if not physically then at least logically). This assumption is supported by the advent of embedded DRAM and 3D memories [8, 91–93] and the deployment of 3D memory in e.g. Intel's 'Knights Landing' Xeon Phi [12]. Thus, we propose a hierarchical cache architecture model and the allocation of the hierarchical caches in three topologies.

A crucial claim in our work is that in order to optimise performance, the communication patterns in multi-threaded shared memory programs will need to exhibit data locality with respect to the memory hierarchy, because the thread scheduler will consider both the load of the cores and the amount of data sharing when placing the threads, see e.g. [94–96]. Our research shows that enforcing memory access locality will improve the latency and throughput. Note in this chapter we use the terms memory block and cache line interchangeably.

5.2 Hierarchical Cache Model (HCM)

In this section, we will describe our proposed hierarchical cache architecture model. In recent years, 3D memory stacking has received great attention [8–11], since it enables much higher memory bandwidth and resolves the memory wall problem of 2D integration. The latest Xeon Phi (Knights Landing) already uses a hybrid form of stacked memory, Micron's Hybrid Memory Cube [12]. 3D memory allows each processor core to have fast and high bandwidth access to the cache banks stacked directly on top of it, using very dense vertical interconnects. The processor core can also access cache banks stacked on the other processors [13–15]. Some of the advantages of 3D stacking are lower latency and higher memory bandwidth. We propose a 3D stacked cache memory hierarchy that has a similar concept to the memory hierarchies of the traditional multiprocessors. The stacked cache memory can be either private or shared [1]. Some research focuses on the techniques for DRAM cache architecture [97]; however, our focus is on the network communications between the cores and the memory, and we abstract away the caches and main memory. Three levels of caches are the most common number used in chips nowadays [39, 40]. IBM zEnterprise 196 (z196) system is already using four levels of cache [41]. However, this is about to change in future many-core systems because as the number of cores in a single chip increases larger caches and more levels are needed [98].

We present a hierarchical cache architecture for three different network topologies, namely the mesh, the concentrated mesh, and the fat quadtree, introduced in the next sections. The caches are stacked on top of one another. Because of their inherent scalability, hierarchical cache architectures are becoming an interesting alternative for many-core systems. Grouping cores and their caches in clusters reduces network congestion by localising traffic among several hierarchical levels, potentially enabling much higher scalability. In our proposed architecture, every four cores are clustered and share a level L2 cache, each 4×4 core is clustered and shares a level L3 cache, and so on. Each cache is shared with the cores below it and private to the other cores. Using $0 < l \leq n$ for the levels of the cache, and assuming $n = log_4(N)$ with N the number of cores, Figure 5.1 shows the proposed hierarchical cache architecture. L1 caches are embedded in the cores; hence, the communication traffic generated from the cores to their L1 private cache is not considered in the evaluation of the communication network. In Figure 5.1, C1, C2, ..., Cn are the individual cores. The caches in our hierarchical cache architecture are totally inclusive. Inclusion means that if a block is cached in any private cache, it is also cached in the shared cache.

5.2.1 Mesh Topology

In a mesh, we place the memory controllers for the hierarchical caches as near as possible to the centre of the cores that share the cache. A router can only represent a single cache level. Figure 5.2 shows an example of the hierarchical cache allocation for an 8×8 mesh. The core can calculate the location of all its hierarchical caches using the following equation, where $\backslash$ represents integer division.

$$4^{l-1}(coreID \setminus 4^{l-1}) + 4^{l-2} - 1,\ 1 \leq l < n \tag{5.1}$$

We use a novel Mesh Memory Routing (MMR) to route between the levels of cache. MMR is a routing mechanism where the routing decision is made at the routers. When a memory request arrives at a cache router, the cache router checks if this cache has a memory block that is shared between the thread in the requester core and the other thread, which is obtained using group or ring clustering. If the shared memory block is in this cache level (cache hit), then a memory block will be sent to the requesting core. If the shared memory block is not in this cache level (cache miss), the request will be sent to the next level cache above until the shared memory block is found. We assume that the last level is the main memory, so if the shared memory block is not in the caches it is requested from the main memory. Then the memory block is returned to the requesting core passing through all the caches from which the request came. The requests are routed through the network between the caches following XY routing.

5.2.2 Concentrated Mesh Topology

The cache placement of the concentrated mesh is similar to the mesh but the cmesh has one less level of cache. In a cmesh, each router represents L2 and might represent another level of cache. The memory traffic routing in cmesh is similar to MMR in the mesh structure of the cmesh then once the flits arrive to the parent router of the core, the router sends the flit to the appropriate core.

5.2.3 Fat Quadtree Topology

The fat quadtree topology reflects our hierarchical cache architecture perfectly, as the leaves represent the cores and the nodes represent the caches. Similarly, four cores share a L2 cache, four L2 caches share a L3 cache, etc.

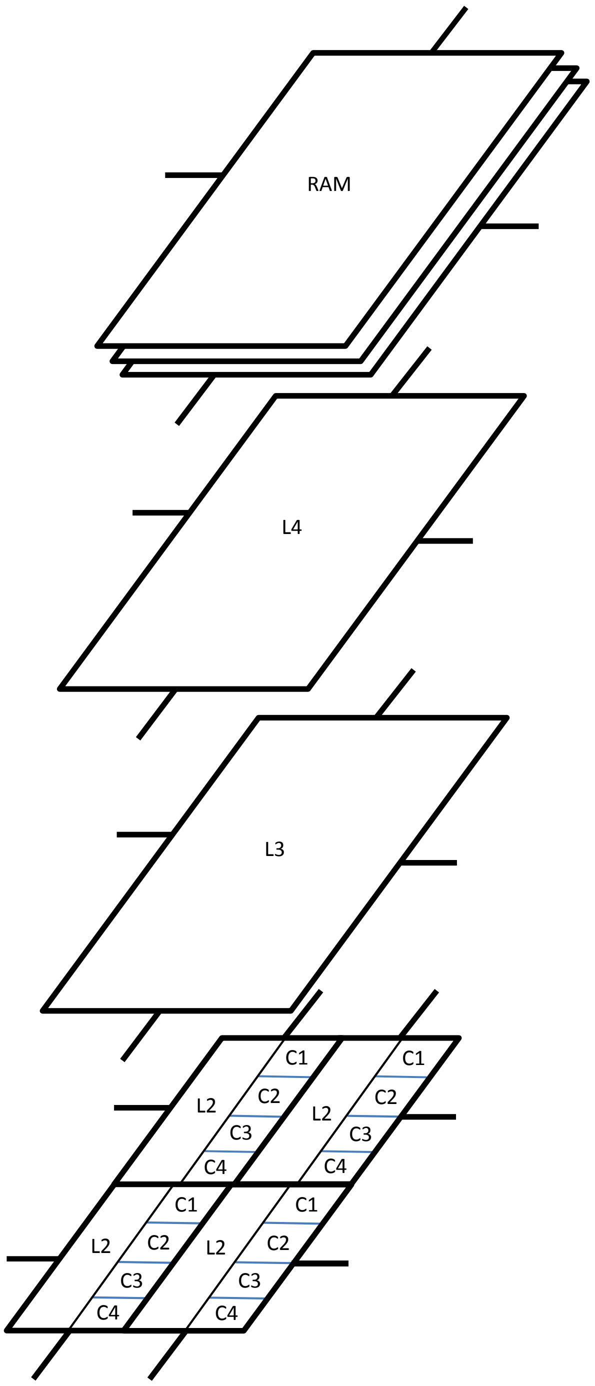


Figure 5.1: The proposed 3D stacked memory ($N = 256$)

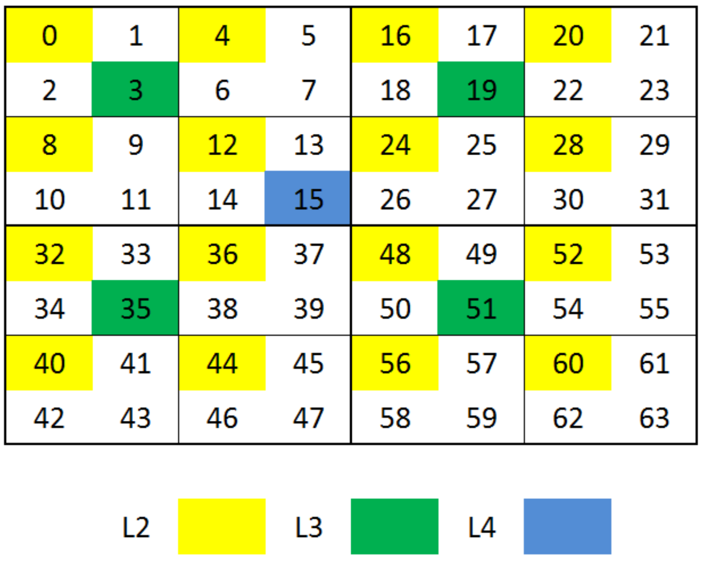


Figure 5.2: Hierarchical cache allocation for an 8×8 mesh

We use a novel routing algorithm in fat quadtree topology to route between the levels of cache. Fat Quadtree Memory Routing (FQMR) is a routing mechanism used to route requests between the levels of cache. When a memory request arrives at a cache router, the cache router checks if this cache has a memory block that is shared between the thread in the requester core and the other thread, which is obtained using group or ring clustering. If the shared memory block is in this cache level (cache hit), then a memory block will be sent to the requesting core. If the shared memory block is not in this cache level (cache miss), the request will be sent to the parent, which is the next cache level above. The requests are routed through the network between the caches following nearest-common ancestor routing. However, when the memory block shared between two threads is found, the memory block is sent back to the requesting core. If the request reaches the highest level router, the main memory, then the memory block is returned to the requesting core, passing through all the caches from which the request came in order to update them.

5.2.4 Links and Buffers per Virtual Channel

In the shared memory traffic model, the flits are required to travel between the caches. The fat quadtree reflects the hierarchical cache architecture perfectly; hence, all the links will be used. However, in the mesh and the cmesh some of the links won't be used at all because the requests travel between the caches using XY routing and the caches are in specific locations. Hence, the same links are always used to travel between the caches. Figure 5.3 shows an example in an 8×8 mesh: around 12.5% of the total links are not used. To compensate for the unused resources in the mesh and the cmesh, more resources (virtual channels and number of buffers) are given to them. Furthermore, the mesh and the cmesh will congest

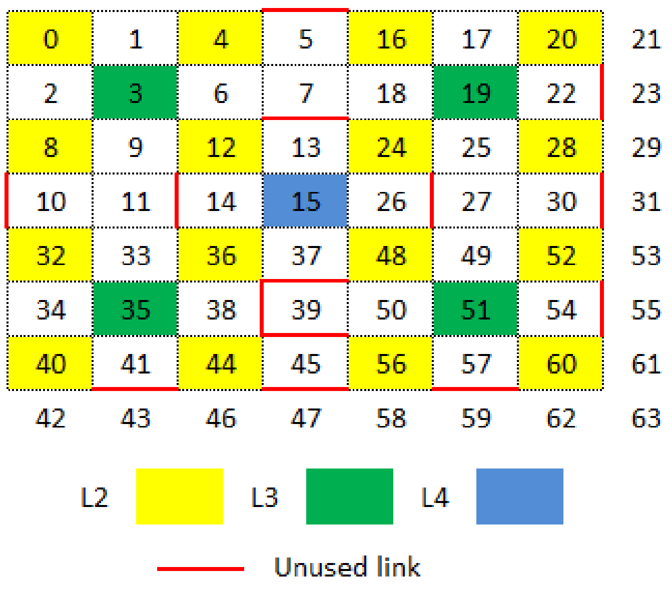


Figure 5.3: 12.5% of the links in an 8×8 mesh are not used

faster compared to the fat quadtree if more resources are not given to them because the cache router and the links between the caches will congest faster.

The fat quadtree has 5120 links while the mesh has 1984 links in a 32×32 network and 22% of those links are not used. Hence, four virtual channels (VC) are assigned to the mesh to compensate for the unused links, giving 6188 utilised links. The cmesh has 480 links with 16% unused links. Therefore, 14 VC are assigned to the cmesh to give 5600 links. Now the mesh and the cmesh have more links than the fat quadtree. Similarly, the number of buffers in a virtual channel in the mesh and the cmesh is now $48\ flit/VC$ and $24\ flit/VC$, respectively.

5.3 Thread Placement Locality Models

We propose simple level-based models to enforce locality in shared memory architecture by placing threads sharing a memory address close to each other. The models are similar to the hierarchical model for point-to-point locality-based traffic mentioned in Section 3.6. However, the traffic in this section is core to cache that shares a memory address between two threads. To model locality of threads placement, we group the cores of the chip and create hierarchical groups to encompass the whole system. Using $0 < l \leq n$ for the levels of the hierarchy, we can express the probability for communication across level-l as:

$$\begin{aligned} p(l) &= (1-\alpha)\alpha^{l-1}, 1 \leq l < n \\ p(n) &= \alpha^{n-1} \end{aligned} \tag{5.2}$$

The parameter α relates to the locality of placing strongly coupled threads. It expresses the probability that a memory request has to travel a certain distance in the hierarchy to a shared memory address with another thread. Lower α means higher locality threads placement: if $\alpha = 0.2$, then according to equation 5.2 there is a 20% chance of the requesting thread finding a shared memory address with another thread in the first level cache, and 20% of that portion in the second level cache, etc. When $\alpha = 1$, it means that most of the threads requesting a memory block will find the shared memory block in the last level cache or main memory.

We use group clustering and ring clustering, which are two different instances of shared memory locality-based thread placement models, to evaluate the performance of the NoC topologies. These are abstract models representing different types of threads placement for different applications. The clustering models are explained in Section 3.6

In general, a thread in a core generates a memory request to a memory block in a cache that is shared with another thread, which is selected based on the clustering model. The memory request results in a cache miss until it reaches the cache that accesses the memory block shared between the threads. The cache then returns the shared memory block to the requested thread. All memory requests generated by the cores are to a shared memory address with other threads, which is the worst case scenario because in realistic systems not all memory requests are to a shared memory block. Most of the data accessed by threads is not shared.

5.4 Simulation Setup

We simulated the different topologies on a 1024-core chip (placed in a regular 32×32 grid) using HNOCS-XT. We model the process-to-cache communication using Poisson-distributed traffic because it has been widely used in both the evaluation of interconnection networks and in cache modelling, see e.g. [99]. The packet length is one flit and the flit size is $64\,Bytes$. The flit size is equal to the size of most of the current system's cache line. The frequency of a modern system is $500\,MHz$. Thus, the channel data rate is $128\,Gbps$ and the bus is $256\,bits$ wide. The clock speed is $2\,ns$. The router delay is $2\,ns$. Four virtual channels are used for the mesh and fourteen for the cmesh as explained in Section 5.2.4, while for the fat quadtree one physical channel is used for the lowest-level links, and it quadruples at each level to simulate a fat quadtree. Hence, the fat quadtree has no virtual channels. The wire de-

Table 5.1: Simulation parameters

Topology	Mesh	Cmesh	Fat quadtree
Number of virtual channels	4	14	0
Wire delay (ns)	17.5	35	$35 \times 2^{l-1}$, $1 \leq l < n$
Flit size (bytes)	64		
Buffer size (flits/VC)	48	24	16
Channel datarate (Gb/s)	128		

lay is proportional to the distance between the routers, so in a fat quadtree it doubles at each level. The threads that share a memory address are selected using different degrees of thread placement localisation, as explained in Section 5.3. Table 5.1 summarises the simulation parameters used in our simulations.

We set the simulation run time as $1\,ms$. In the simulation, all threads in all cores generate memory requests at the same time. A thread requests a memory block only if the memory request in the core queue is sent; thus, there will be no dropped memory requests.

5.5 Simulation Results and Discussion

We evaluated the performance of the three topologies as a function of the memory access generation rate. The miss rate α, where $\alpha = 1$, means that the threads are replaced the furthest and the main memory will be hit, and $\alpha = 0$ means the first level cache will be 100% hit. Memory access is the operation of reading or writing stored information in memory. The thread sends a memory access request when it needs to read or write to memory. We assume that all the memory blocks are shared between two threads. Here, latency is the latency of the network, not taking into account the waiting time of a packet at the transmitting core when the link is busy.

5.5.1 Effect of Locality on Latency

For group clustering, Figure 5.4 shows that the fat quadtree performs best. The cmesh and the fat quadtree have lower latencies when $(0 \leq \alpha \leq 0.1)$. The latencies start to increase as α becomes higher. The cmesh congests faster than the fat quadtree because it has fewer routers. The mesh has high latencies although it has more routers than both the fat quadtree and the cmesh. This is because in shared memory traffic, where the requests travel between the caches, the path of the requests is deterministic; hence, some routers have more traffic while others might not receive any traffic. Moreover, most of the links that connect between the caches are nearly 100% utilized. Some of the cores do not receive any flits due to the congestion, so we set the latency penalty for those cores to the simulation time $1\,ms$.

For ring clustering, Figure 5.5 illustrates that the mesh and the cmesh have high latencies as they congest faster and they perform worse than the fat quadtree. The fat quadtree has low latencies. This is because in the mesh and the cmesh the memory request traffic always travels on the same path between the caches, and so the network gets congested.

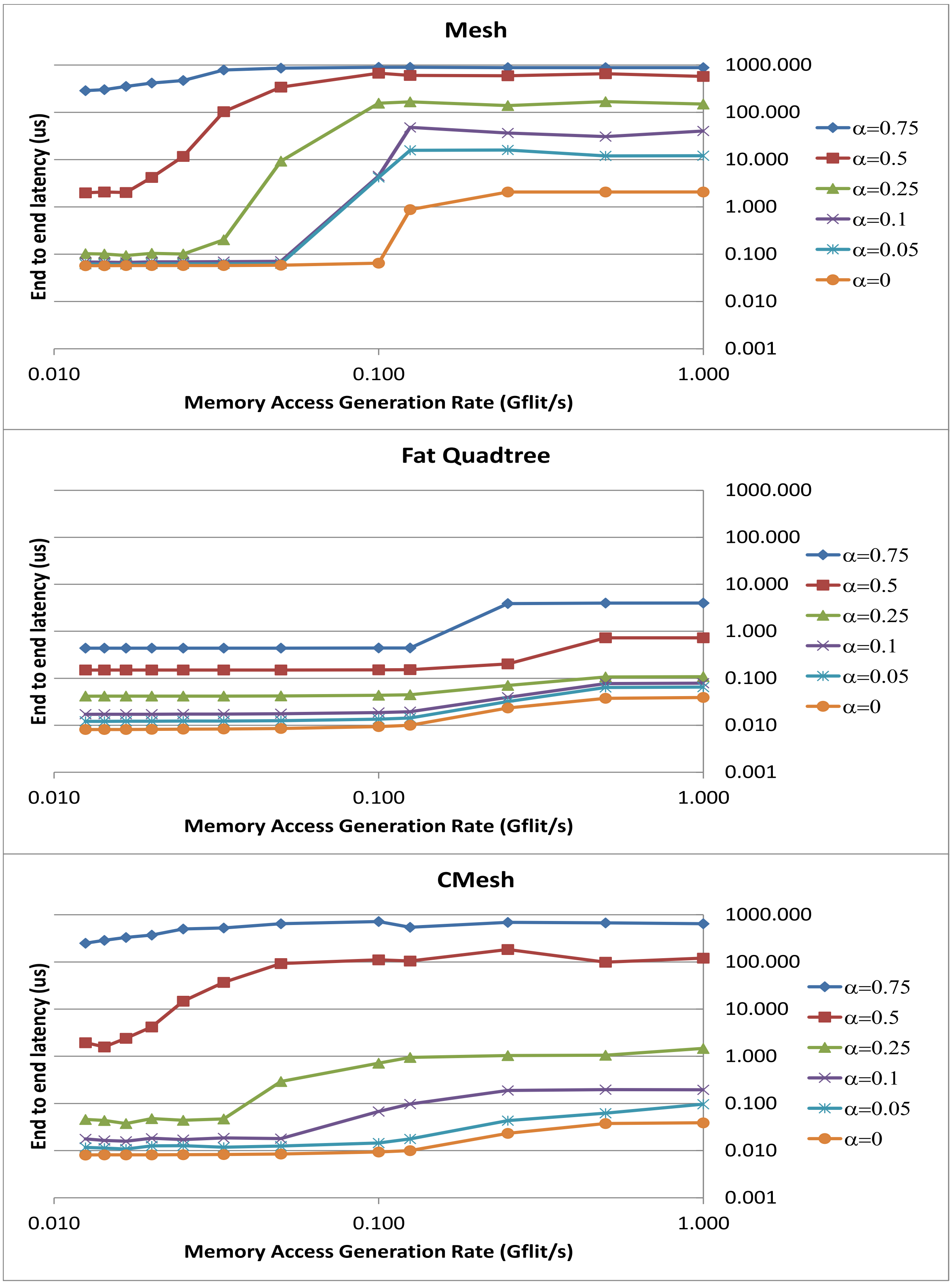


Figure 5.4: Memory requests' latency for group clustering

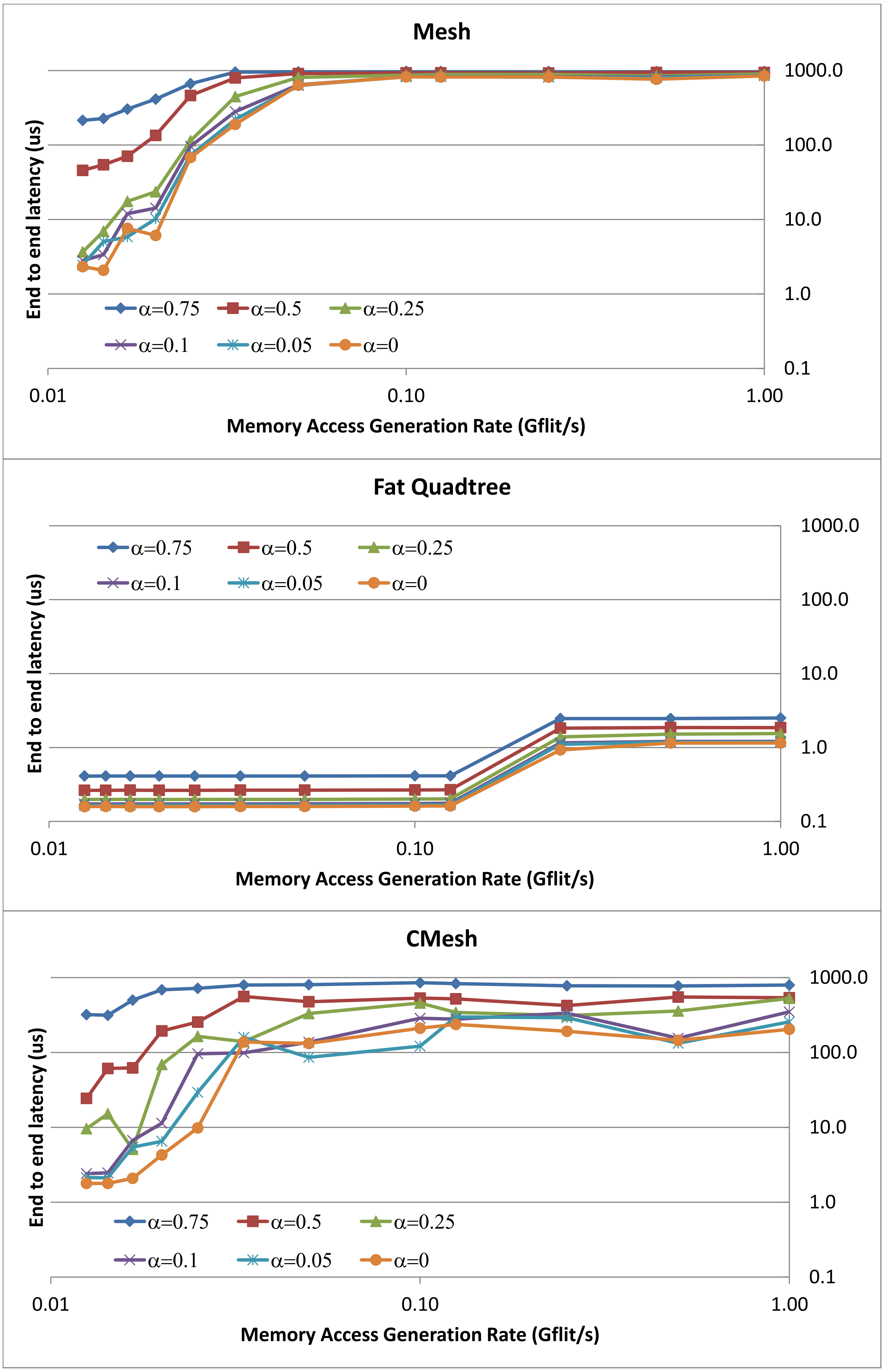
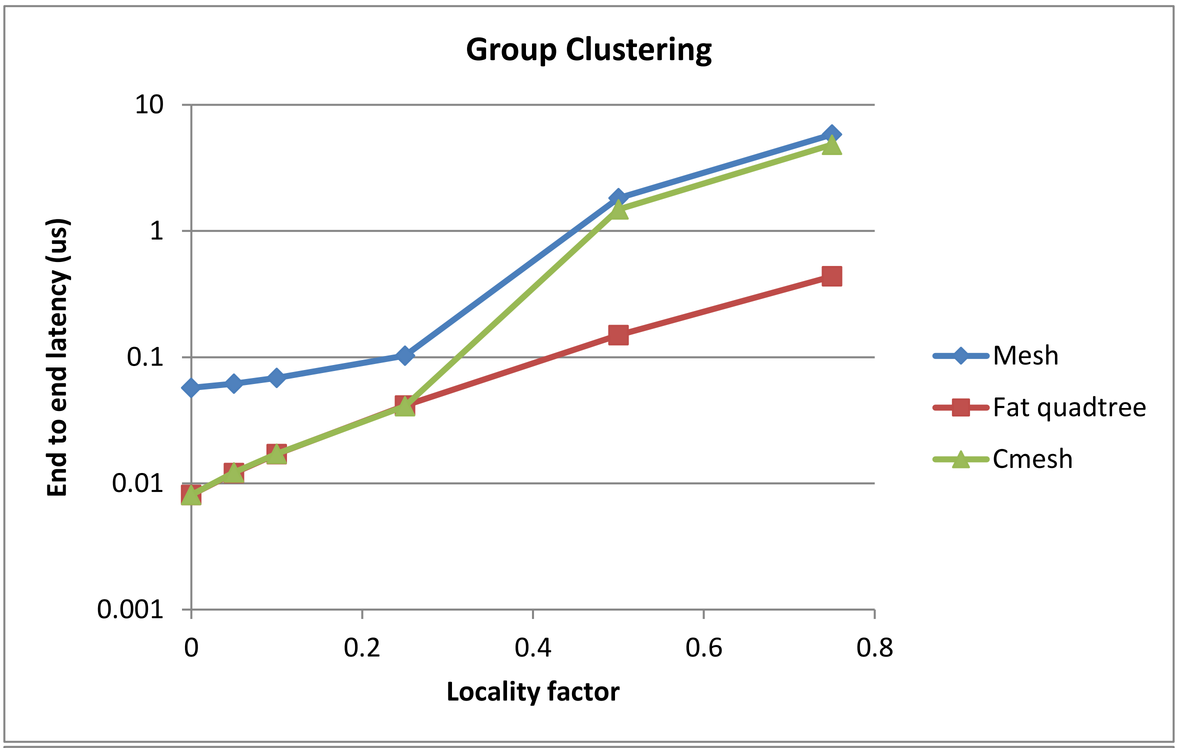


Figure 5.5: Memory requests' latency for ring clustering



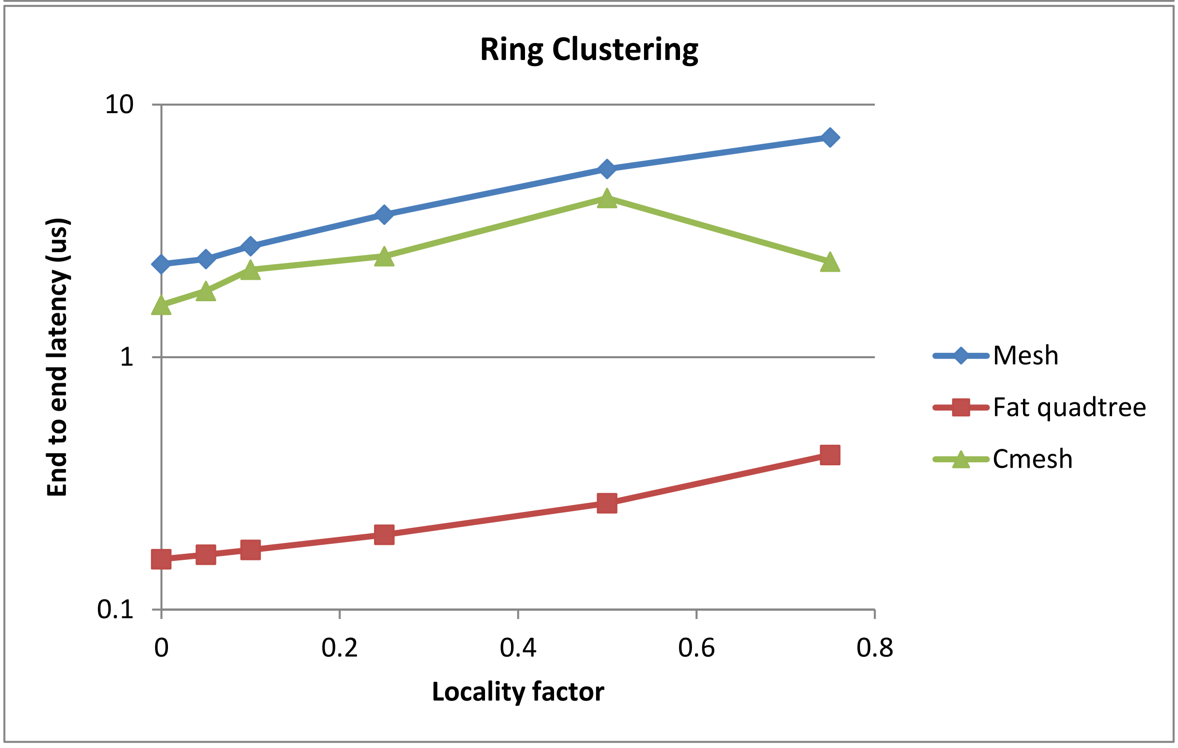


Figure 5.6: Group and ring clustering (1: no locality, 0: total locality)

Overall, group clustering results in lower latencies than ring clustering. This is an important result for the placement of threads in group neighbours that match the cache architecture. Figure 5.6 shows the latencies of the topologies when the network is not congested and packets are generated every $80\,ns$ for different α parameter. In both models, the fat quadtree performs best even when the threads are placed further apart.

Figure 5.7 shows an example for a ring clustering case. Core 27 sends a memory request to a shared memory with one of these cores: 48, 49, or 52 and back to core 27. It illustrates the path that a memory request traveling from core 27 has to pass through until it reaches a cache shared with cores 48, 49, and 52, which is L4 cache. The request has to go to L2 (router 24), then L3 (router 19), then L4 (router 15), then pass back through all the caches to update it and acknowledge it to the requesting core 27.

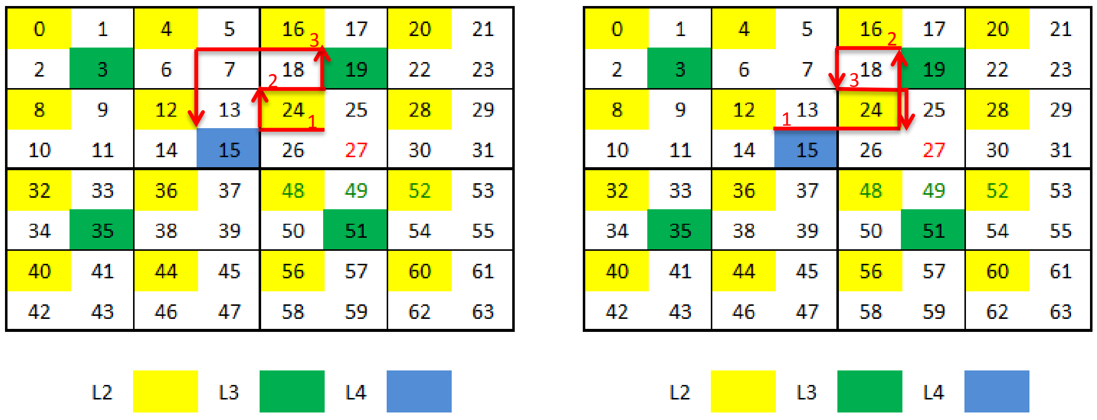


Figure 5.7: Example of a memory request in an 8×8 mesh ring clustering (core 27 sends a memory request to a shared memory with cores 18, 19, or 52 and back)

5.5.2 Effect of Locality on Throughput

For group clustering, Figure 5.8, we observe that the fat quadtree has the highest throughput in all the cases. As the locality decreases the throughput decreases, because when locality decreases, the requests have to travel further through the caches. Therefore, it takes a longer time to reach the common cache and get back to the requester, which leads to longer delays and a congested network. In the case of $(0 \leq \alpha \leq 0.1)$, the fat quadtree and the cmesh have nearly similar throughput because of the similarity of the lower level structure of both topologies. In the case of $(0.25 \leq \alpha \leq 0.75)$, the cmesh throughput starts to decrease compared to the fat quadtree due to more requests traveling to higher levels of caches, which have the same structure as the mesh. The mesh has the worst throughput of all. For ring clustering, Figure 5.9, the fat quadtree has higher throughput compared to the cmesh which has higher throughput than the mesh.

We now compare the fat quadtree with the existing systems today. Overall, the memory bandwidth of our fat quadtree system in high memory access generation rate is around $15\,TB/S$ in group clustering and $13\,TB/S$ in ring clustering. In the present systems, the memory bandwidth for NVIDIA's Kepler K40 GPU and Intel's Xeon Phi 7120P are $0.28\,TB/S$ and $0.34\,TB/S$, respectively. This shows that our 2023 system has much higher throughput, taking into account that the wire delay is much higher and our system has all shared memory communication, which is the worst case.

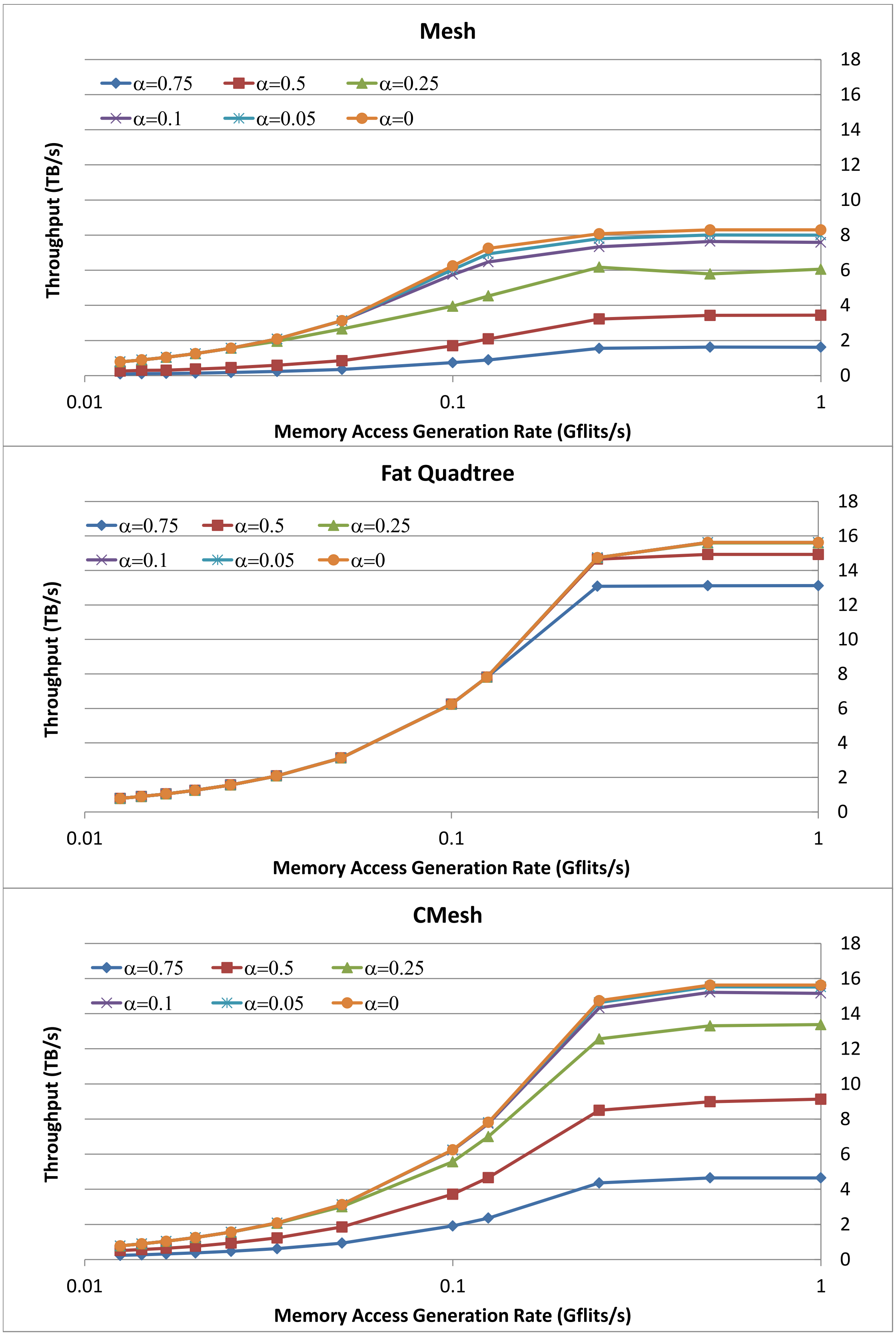


Figure 5.8: Throughput for group clustering

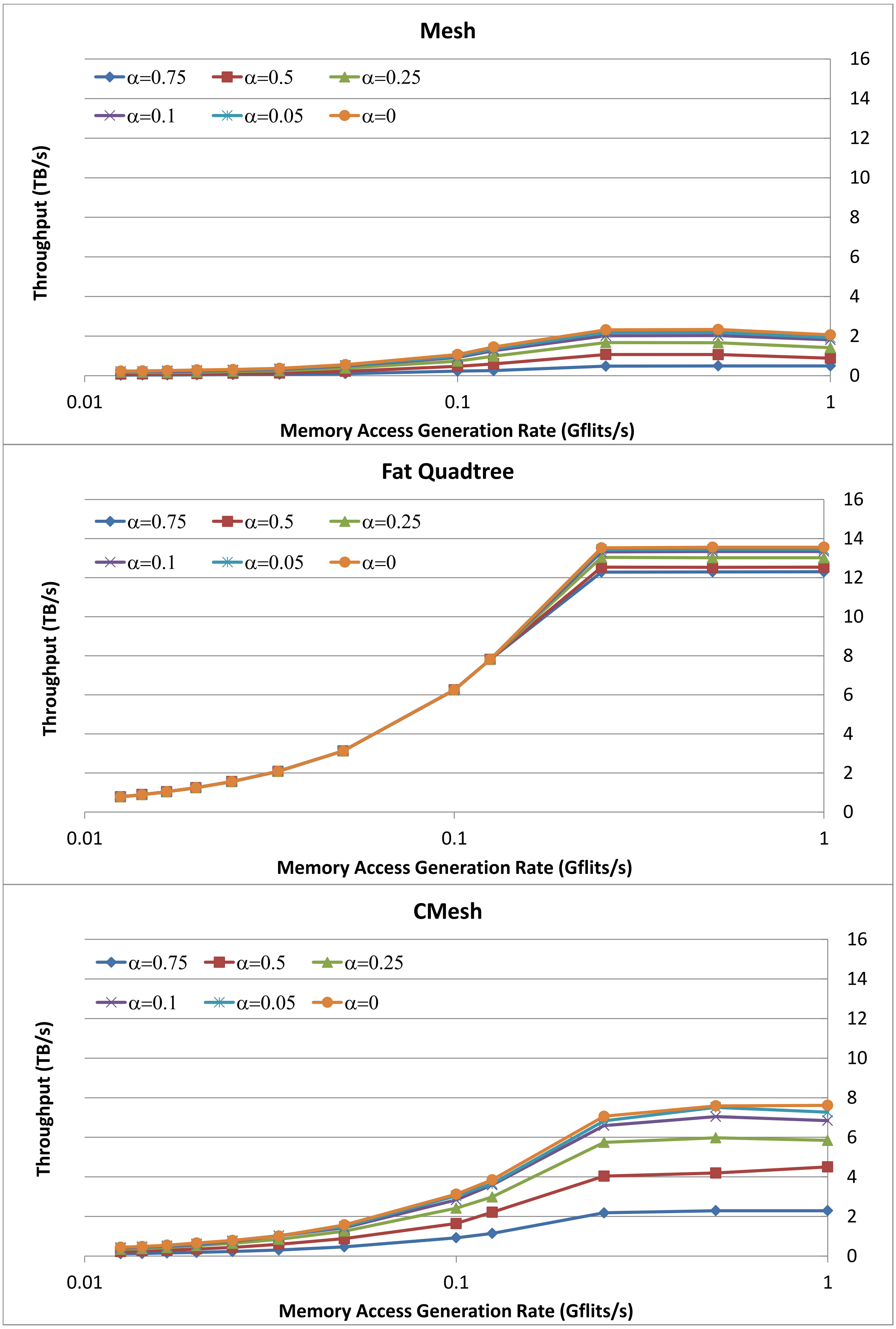


Figure 5.9: Throughput for ring clustering

5.6 Summary

In this chapter, we have proposed a hierarchical cache model and suggested the cache placements in three topologies. Moreover, we have investigated the overhead and performance of flat (mesh, cmesh) and scale-invariant (fat quadtree) NoC topologies for future many-core systems with thousands of cores using two different models of localised threads placement in shared memory systems, group clustering, and ring clustering. We have shown that the distance between the threads sharing a memory block and the clustering model strongly affect the performance of the network. Scale-invariant topologies, such as the fat quadtree, perform better than flat ones because their structure matches the hierarchical cache architecture. In addition, the fat quadtree has a direct link between the levels of cache, although the wire delay increases as the request travels up the tree. Our results clearly show the importance of localised threads placement for very large many-core systems. The model we have discussed in this chapter is without cache coherency, and we assume that the programmer handles the cache coherency. However, in the next chapter we will integrate the cache coherency in our hierarchical cache model to make it easier for the programmers to use the chip. Will cache coherency cause the system to crash as the conventional wisdom suggests?

Chapter 6

Cache Coherency for Many-Core Systems

This chapter integrates cache coherency into a hierarchical cache model using shared memory architecture. It presents the cache coherency protocol we suggest for many-core systems. It shows the importance of cache coherency in a shared memory architecture. In addition, this chapter describes the hierarchical tracking of shares. It provides the cost model of the cache coherency protocol in terms of the cache size and the communication performance in the shared memory hierarchical cache model. It provides the simulation setup, the results, and the discussion of the simulation results. The chapter shows that the directory-based cache coherency scales, and that its effects on system performance are small when using the hierarchical cache model. We show that cache coherency scales using the suggested cache coherency protocol and hierarchical cache model.

6.1 Cache Coherency

Cache coherency is very important in shared memory many-core systems. It ensures the consistency of shared data that is stored in multiple caches. The conventional wisdom has been that cache coherence could not scale because of exploding storage and interconnection network traffic requirements, and concerns over latency and energy consumption. However, Martin et al. [52] discuss some techniques that can be used to scale the cache coherency. They examine five potential concerns when scaling on-chip coherence. First, regarding the traffic on the on-chip interconnection network, they show that it scales when precisely tracking sharers. Secondly, regarding storage costs for tracking sharers, they show that a hierarchy combined with inclusion enables efficient scaling of the storage cost for exact encoding of sharers. Thirdly, for inefficiencies caused by maintaining inclusion, they find that using chip architects to design a system with an inclusive shared cache with negligible recall rate can efficiently embed the tracking state in the shared cache. Fourthly, latency of cache misses is shown to be tolerable, as misses to actively shared blocks have greater latency than other misses. Finally, the energy overhead analysis shows that based on the traffic and storage scalability analyses, the energy overhead of coherence will not increase with the number of cores.

According to the findings of Martin et al. [52] we can assume that the traffic overhead of the cache coherency traffic will be negligible. Furthermore, Gustavo et al. [100] show that the amount of application data in the NoC are much larger than the amount of cache coherence data for almost all cache sizes. In this chapter, building on the arguments of Martin et al. [52], we assume a shared memory model of computation. Our proposed hierarchical cache architecture, which was explained in Chapter 5, provides a good baseline for a hierarchical cache architecture with cache coherency. We use a directory-based protocol and hierarchical tracking of sharers on our hierarchical cache architecture. We assume that the caches and the main memory are abstract. We focus on the communication side, the messages generated using cache coherency protocol, and its overhead on the network.

6.2 Cache Coherency Protocol

There are two cache coherency protocol methods of implementation, which are the snooping and the directory-based protocols [53, 57]. We use a directory-based cache coherency protocol because it scales and generates less coherency traffic. Directory-based cache coherency protocol assumes a shared memory space, which is physically distributed. The idea of it is to implement a directory that keeps track of where each copy of a memory block is cached and its state in each cache [56]. The snooping protocol keeps the state of the block in the cache only. Cores must consult the directory before loading memory blocks from the main memory to the caches. When a memory block in the cache is updated (written), the directory is consulted to either update or invalidate the other cached copies. This eliminates the overhead of broadcasting; hence, it scales.

The directory keeps track of where each memory block is stored in the cache. Accordingly, a memory block can be in one of three states. It can be un-cached and no core has it; in this case, the memory block should be requested from the main memory. It can be shared and clean, when it is cached in one or more caches and the block is up-to-date with the data in the main memory. It can be exclusive and dirty, when one processor owns the data and is out of date with the data in the main memory. Note in this chapter we use the terms memory block and cache line interchangeably.

There are different ways to implement a directory structure in many-core systems:

- Full-map directories share a given piece of data simultaneously with all the cores in the chip. Consequently, they have to maintain an enormous amount of status information [56]. Precisely, each block in main memory must maintain a complete list of all the caches in the system. Therefore, the amount of data required to store the state of each memory block scales linearly with the number of cores [101].

- Duplicate-tag-based directories mirror the organisation of the private-cache tags. This ensures that there is always enough space in the directory to track all cached blocks [102]. The duplicate-tag associativity must equal the product of the cache associativity and the number of caches, which results in large associative directories [102]. Because of the large associative directories, duplicate-tag-based directories are non-scalable. The Xeon Phi processor uses this type of directory-based protocol [33].

- Sparse directories' goal is trying to reduce the space of directory storage by reducing directory associativity. When sparse directory space overflows, it experiences set conflicts and forced evictions of cached blocks that cannot be simultaneously tracked by the directory [101].

- In-cache directories extend an inclusive shared cache's tags with the sharer information, implicitly saving directory tag storage, but grossly over-provisioning the sharer storage because the number of tags in the lower-level cache greatly exceeds the number of tracked blocks in the private caches [102]. The SGI Origin2000 multiprocessor system [103] and Tilera Tile64 [2] are real architectures that implement in-cache directory. In-cache directory integrates the directory state with the cache tags, thereby avoiding a separate directory either in DRAM or on-chip SRAM. The tag overhead can become huge in many-core systems with a bit for each sharer. However, to reduce tracking sharers' overhead, a hierarchy combined with inclusion enables efficient scaling of the storage cost for exact encoding of sharers [52].

In our directory-based cache coherency protocol, we use an in-cache directory that extends an inclusive shared cache's tags with sharers' information and distributes directory among cores. Because of the hierarchical nature of our cache architecture, each memory block requires $4\,bits$ to track sharers and maintain the inclusiveness of the caches. Figure 6.1 shows how tracking sharers is done in our hierarchical cache model. Since every cache level has four lower level caches, we need only 1 bit for each lower level cache to know where the shared memory block is. In Figure 6.1, the highest level memory block has the bits 0010 as the cache tag. Thus, the block is shared and the third lower level (level 2) cache has a copy of the block. It has a 0100 cache tag, which indicates that its second lower level (level 1) cache has a copy.

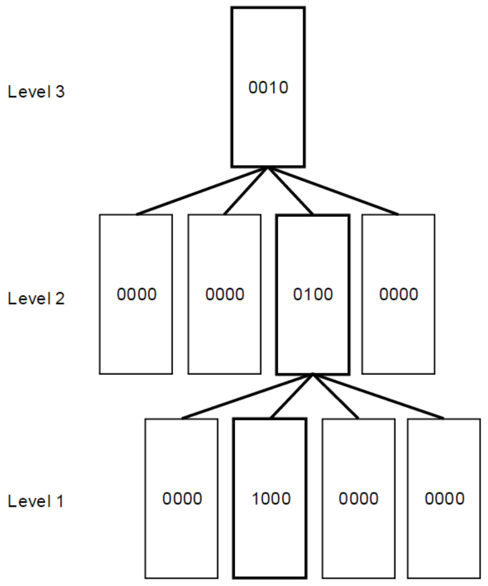


Figure 6.1: Tracking sharers

Although using hierarchical tracking sharers requires additional layers of cache lookup, it has two key advantages for coherence. First, it reduces the number of coherence messages. For example, if a number of cores shares a block, and one of them writes to the block on the shared last level cache, only one invalidation message can be triggered for each cluster that shares the block. The second advantage is that it enforces inclusion at each level, which reduces the storage cost of coherence. Because we are using an exact tracking of sharers, this allows for scalable communication.

There are two ways in which updates to cached memory can be made. A write-through policy updates to main memory directly every time a change is made in a cache line. However, a write-back policy is able to avoid these expensive writes to main memory by making these updates in the caches. When data in a write-back cache needs updating, its value in the cache is updated and the cache line state changes to dirty. It is only when the cache line is evicted that the write actually goes out, and a main-memory transaction occurs. For our directory-based cache coherency protocol we use a write-back policy.

Figure 6.2 shows the flow chart of our directory-based cache coherency protocol. For a read memory request, the request goes to the first cache level. If the memory block exists in the cache and is a read hit, then it is sent to the requested core. If the memory block does not exist in the cache and is a read miss, then the request goes to the next level cache until it is a hit, assuming the last level is the main memory. If the request is a hit in a higher cache level and the cache line is clean and shared, then the cache line is copied to the lower cache levels and to the requested core. However, if the cache line is dirty and exclusive, then it is updated in the main memory and the state of the cache line is changed to clean and shared.

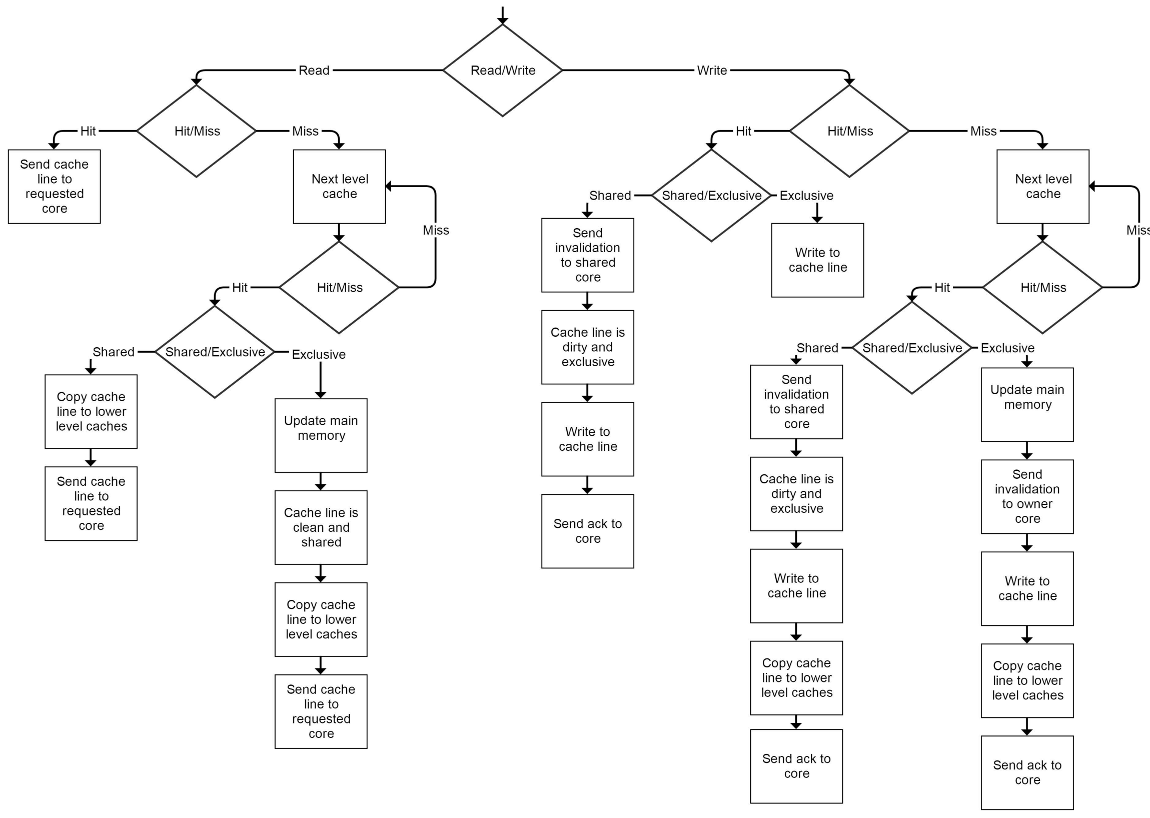

Read/Write
Read
Write
Hit/Miss
Hit
Miss
Send cache line to requested core
Next level cache
Miss
Hit/Miss
Hit
Shared/Exclusive
Shared
Exclusive
Copy cache line to lower level caches
Send cache line to requested core
Update main memory
Cache line is clean and shared
Copy cache line to lower level caches
Send cache line to requested core
Hit/Miss
Hit
Miss
Shared/Exclusive
Shared
Exclusive
Send invalidation to shared core
Cache line is dirty and exclusive
Write to cache line
Send ack to core
Write to cache line
Next level cache
Miss
Hit/Miss
Hit
Shared/Exclusive
Shared
Exclusive
Send invalidation to shared core
Cache line is dirty and exclusive
Write to cache line
Copy cache line to lower level caches
Send ack to core
Update main memory
Send invalidation to owner core
Write to cache line
Copy cache line to lower level caches
Send ack to core

Figure 6.2: Directory-based protocol chart

After that, the cache line is copied to the lower level caches and to the requested core.

For a write memory request, the request goes to the first cache level. If it is a write hit and the memory block is clean and shared, then invalidations are sent to the cores sharing the memory block. The memory block is updated and its new state is dirty and exclusive. An acknowledgment is sent back to the core. If the memory block is exclusive, then it is updated and an acknowledgment is sent back to the core. However, if it is a write miss, then the request goes to the next cache level until it is a hit, assuming the last level is the main memory. If the request is a hit in a higher cache level and the cache line is clean and shared, then invalidations are sent to the cores sharing the memory block. The memory block is updated and its new state is dirty and exclusive. The cache line is copied to the lower cache levels and an acknowledgment is sent back to the core. If it is exclusive, then it is updated to the main memory and invalidations are sent to the cores sharing the memory block. After that, the cache line is copied to the lower cache levels and an acknowledgment is sent back to the core.

6.3 Cost Model

Since the directory is part of the cache, we need to compute the directory overhead on the cache. The directory stores the status of all the blocks in the memory whether it is clean or dirty. This requires $1\,bit/memoryblock$. In addition, it stores the cores that have copies of the memory block to keep track of sharers. Because of the hierarchical structure of our model we only need $4\,bits$ for each memory block as explained in Section 6.2. Consequently, $5\,bits$ for each memory block are required by the directory. The memory block size is $64\,Bytes$; therefore, the total directory overhead is 0.97% for each memory block.

6.4 Simulation Setup

We simulate the many-core system on a 1024-core chip for different topologies using HNOCS-XT. We use the same thread-to-thread communication used in Section 5.4; however, the cache coherency traffic is added. Threads running on many-core systems use shared caches to communicate and share data with each other. We use the same simulation parameters and setups used in Section 5.4. To simulate the cache coherency protocol as explained in Section 6.2, we created a memory model in HNOCS-XT to simulate the main memory and the caches. According to the protocol, the memory model generates cache coherency traffic and eviction messages. In addition, it uses the following parameters: the read/write rate (P_{rw}), the shared rate (P_s), and the eviction rate (P_e). The read/write rate (P_{rw}) defines the read and write requests' ratio. The shared rate (P_s) defines the ratio of reads and writes to shared data, as not all reads and writes are to shared data. The eviction rate (P_e) defines the ratio of generating eviction messages. For an application running on a shared memory architecture, not all instructions are memory accesses. In addition, not all reads and writes are communicating data. In this research, we focus on reads and writes that send data to the network to access memory.

We use SPLASH-2 [104] and PARSEC [105] benchmarks to select the proper rates to use in the simulation. SPLASH-2 is a well-known benchmark suite containing a variety of high performance computing and graphics applications. PARSEC is a more recent benchmark suite that has a wider variety of applications. In SPLASH-2 and PARSEC benchmarks applications, memory accesses are between 27.83% and 35.99% of all instructions [105]. The worst application in terms of a miss rate is *canneal,* which has a miss rate of 3.18% with $4\,MB$ caches [105]. However, the *blackscholes* application has the lowest miss rate of all applications of 0.01% with $4\,MB$ caches. The worst case for shared writes is in the *Lu* program in SPLASH-2, with 27.40% shared writes with $8\,MB$ caches [105]. As the cache becomes large enough to keep the shared part of the working set, the miss rate decreases and the shared rate increases. In terms of the memory access generation rate, *streamcluster* is a typical application in PARSEC, which has a memory access of $0.875\,B/instruction$ [105]. This is equivalent to $0.0068\,Gflits/s$ in our system memory access generation rate.

In our model, the miss rate is equivalent to our locality parameter α. Thus, we assume that $\alpha = 0.05$ which is equivalent to 95% locality or to 5% miss rate. In most of our simulation cases, we use 50% shared writes.

6.5 Results and Discussion

We evaluate the performance of the three topologies as a function of the memory access generation rate, the read/write rate (P_{rw}), the shared rate (P_s), and the eviction rate (P_e). The miss rate is fixed and equal to $\alpha = 0.05$. Group clustering (G) and ring clustering (R) are the two clustering methods we use to choose the core that has threads sharing a memory block. There are two standard cases to compare the results with. The first case is the Hierarchical Cache Model (HCM) with no cache coherency traffic in Chapter 5, and the other case is the cache coherency case when all the parameters are equal to zero $(P_{rw} = 0,\ P_s = 0,\ P_e = 0)$. The second case should give nearly the same results as the HCM case with a slight increase in latency. No cache coherency traffic is generated as all the memory requests are read requests, and the slight increase in latency is caused by the computation delay in the memory model.

6.5.1 Effect of Cache Coherency Traffic on Latency

For group clustering, Figure 6.3, the fat quadtree and the concentrated mesh perform better than the mesh. However, the fat quadtree saturates faster than the concentrated mesh. Both topologies have the same low level architecture, and when $\alpha = 0.05$ it means that 95 of the memory requests will be a hit on the first level cache. Moreover, the 5% miss requests go to a higher cache level, which makes the fat quadtree saturate faster. Because of this, the fat quadtree has to have longer links with higher wire delay in order to go to the next level compared to the concentrated mesh. This leads to the question of whether the concentrated mesh will still perform better than the fat quadtree if the number of cores increases. The mesh performs worse because many packets travel between the caches using the same links; hence, the links are fully utilised, which slows the traffic and increases the latency. Note that the mesh saturates on the case $(P_{rw} = 0.2,\ P_s = 0.2,\ P_e = 0.2)$, which is not even the worst case scenario. Figure 6.5 illustrates the heatmap for the mesh when it is saturated. It shows that only a small number of cores (red and yellow cores) have a very high latency, which gives a higher total average latency. The green coloured cores are the most common in the chip and they have low latencies. This is because the mesh uses the same links to move packets between the caches, which leads to higher utilisation of some links. Figure 6.6 describes the frequency of latency for all the packets received by three different cores from each colour group (core[66], core[230], core[239]).

For ring clustering, Figure 6.4, the fat quadtree performs best, although as the memory access generation rate increases it starts to saturate. The fat quadtree has an advantage over the mesh and the concentrated mesh because it has direct links between the caches, unlike the flat topologies that use the same links to go through the caches, which makes the links utilise faster.

Now comparing the *streamcluster* application in our system in group clustering, we see that the mesh is already saturated but the application has a low latency in the concentrated mesh and the fat quadtree. However, the application has a higher latency in the ring clustering.

When taking a closer look at the ring fat quadtree, the standard case $(P_{rw} = 0,\ P_s = 0,\ P_e = 0)$ is saturated differently than the HCM-No-Coherency case. The heatmap of the ring fat quadtree in Figure 6.7 shows that it saturates faster when the memory access generation rate increases. The heatmap shows that the cores that have higher latencies are near the edge of the group. Since this is ring clustering, this behaviour is expected because in a fat quadtree neighbouring cores might not have a direct common ancestor, and so the packet has to travel further up the tree.

For the cmesh in ring clustering, although the memory access generation increases the latency does not increase saturation much. In Figure 6.8, the heatmaps show the latency of each core when the access generation rate is very low, the bottom figure, and when the access generation rate is high, the figure on top. More cores get higher latency as the generation rate increases, but this is not reflected in the average latency in Figure 6.4 because some cores have a very huge latency; even if many cores have lower latency, it does not affect the average.

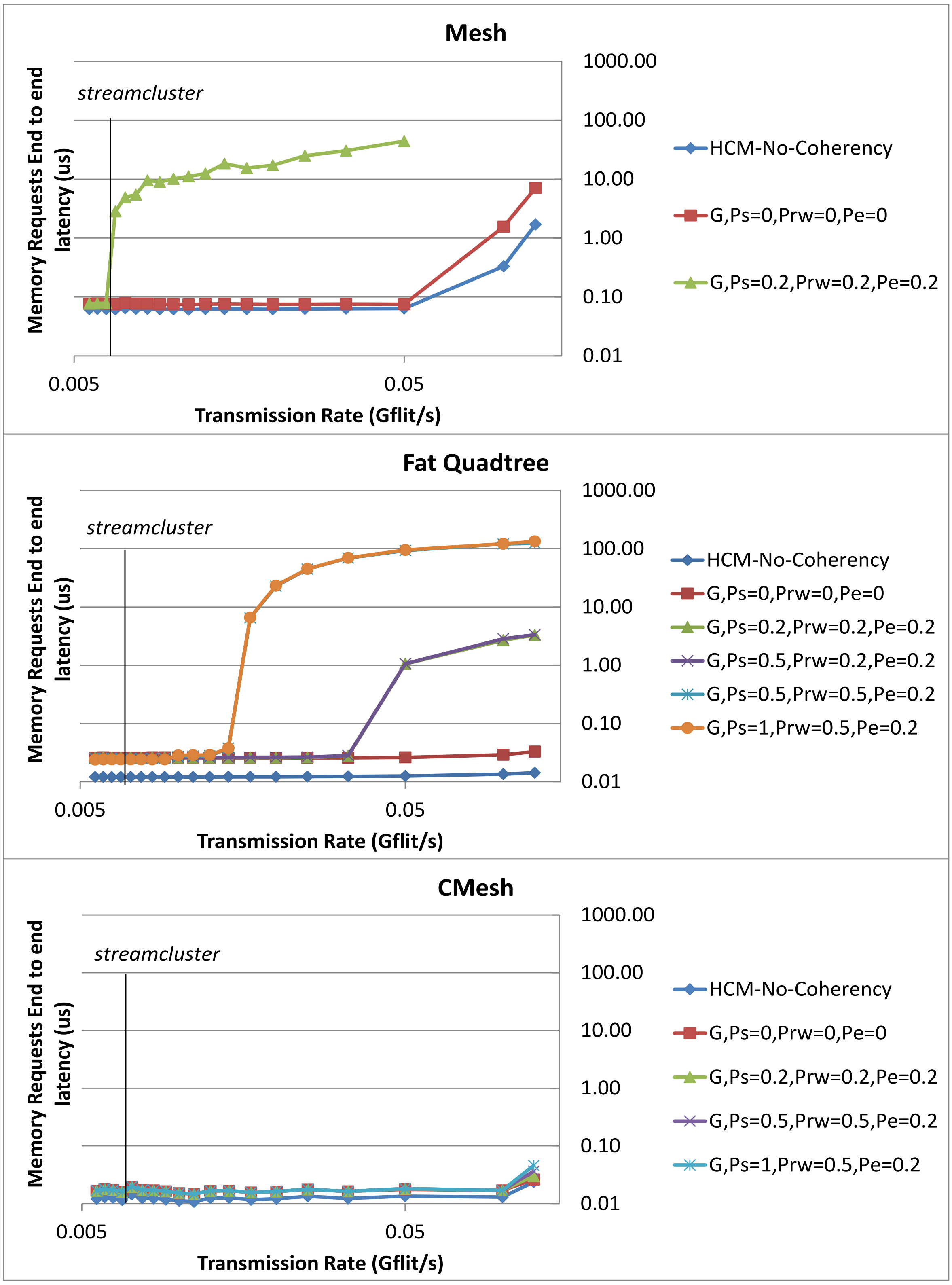


Figure 6.3: Memory requests' latency for group clustering (G: group clustering, P_s: the shared rate, P_{rw}: the read/write rate, and P_e: the eviction rate)

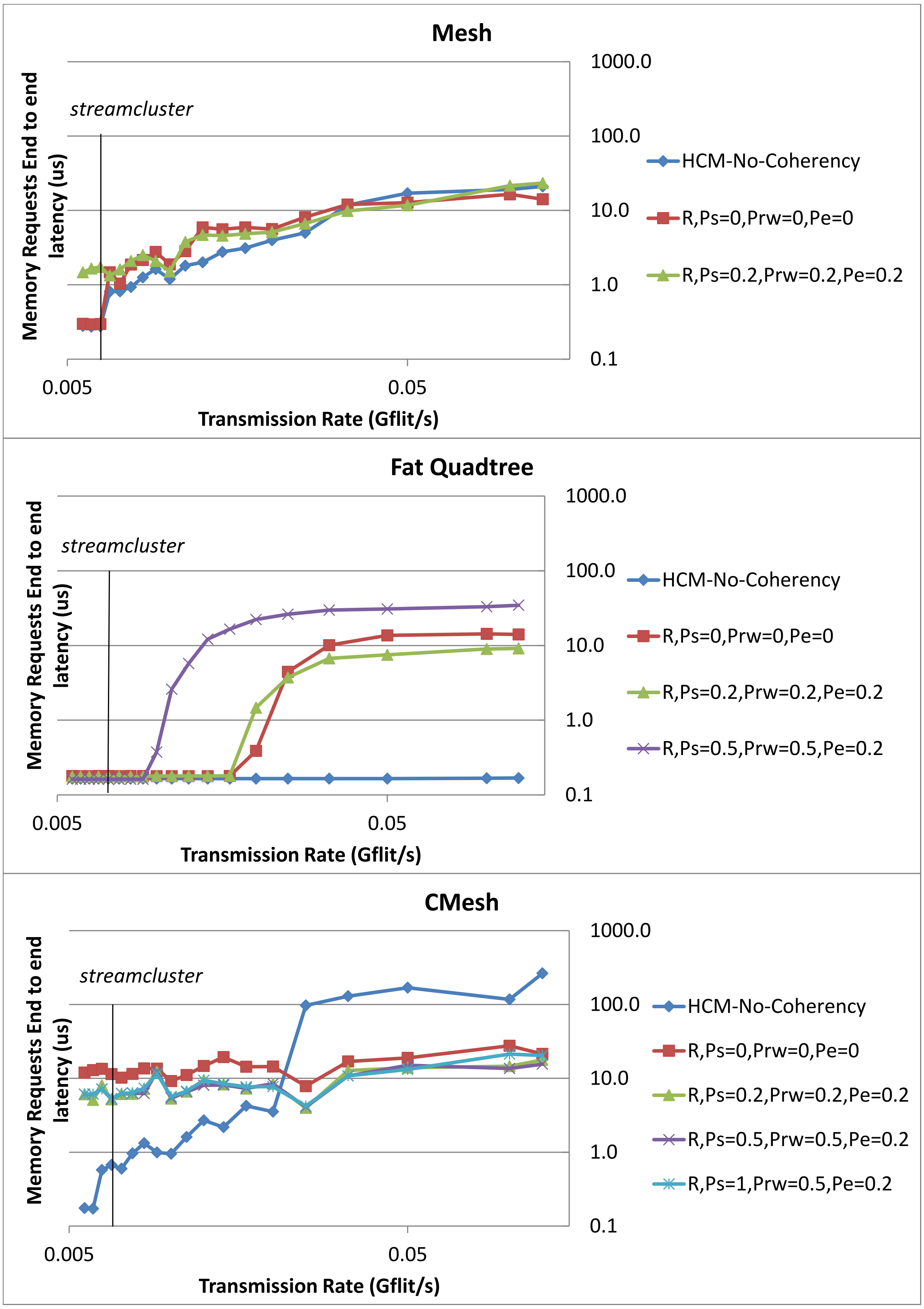


Figure 6.4: Memory requests' latency for ring clustering (R: ring clustering, P_s: the shared rate, P_{rw}: the read/write rate, and P_e: the eviction rate)

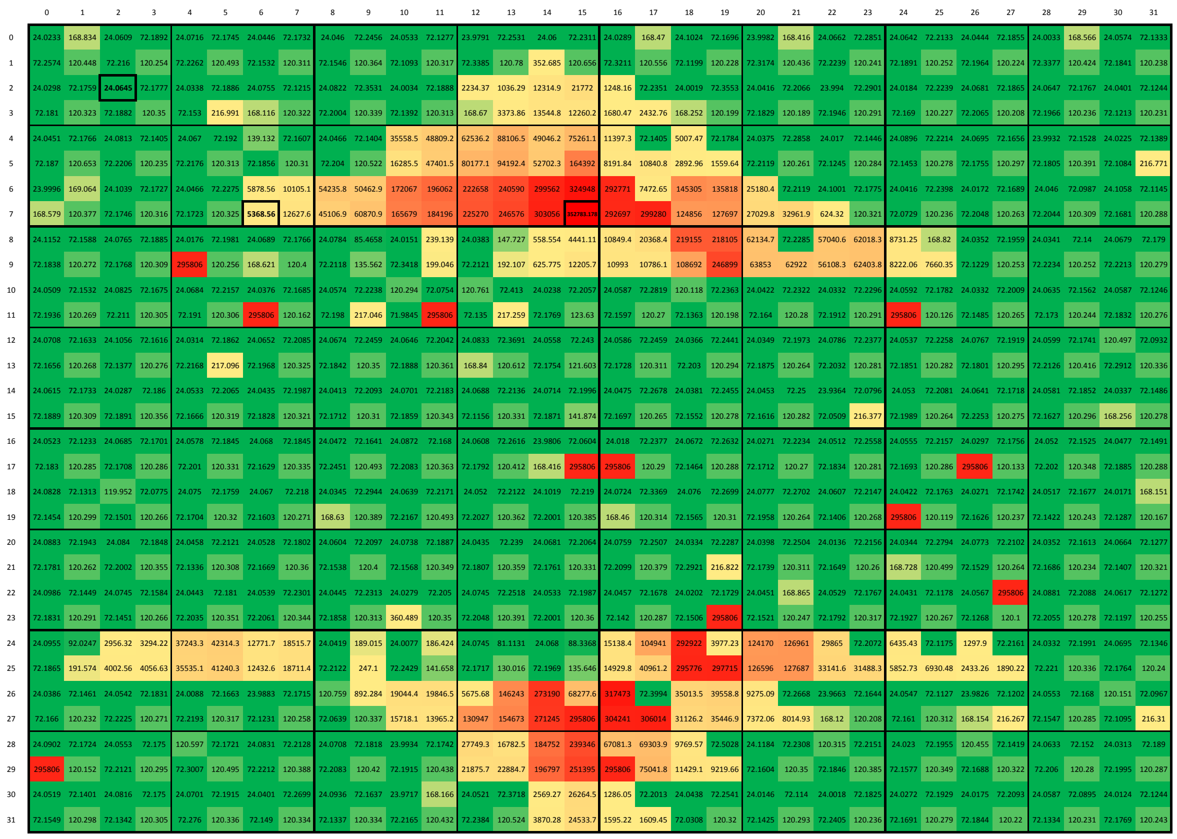


Figure 6.5: Mesh heatmap for group clustering

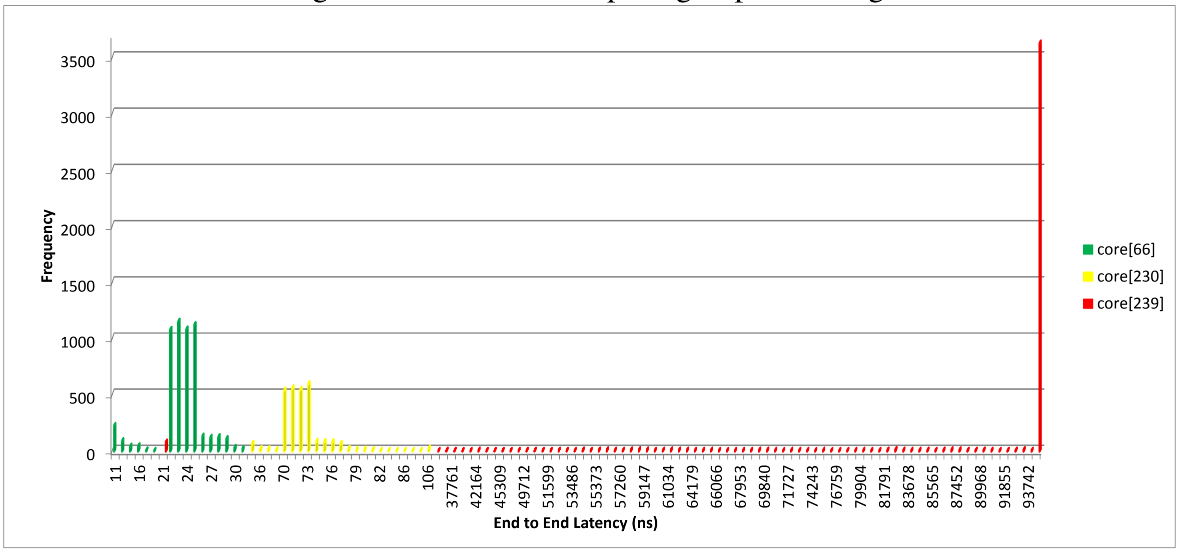
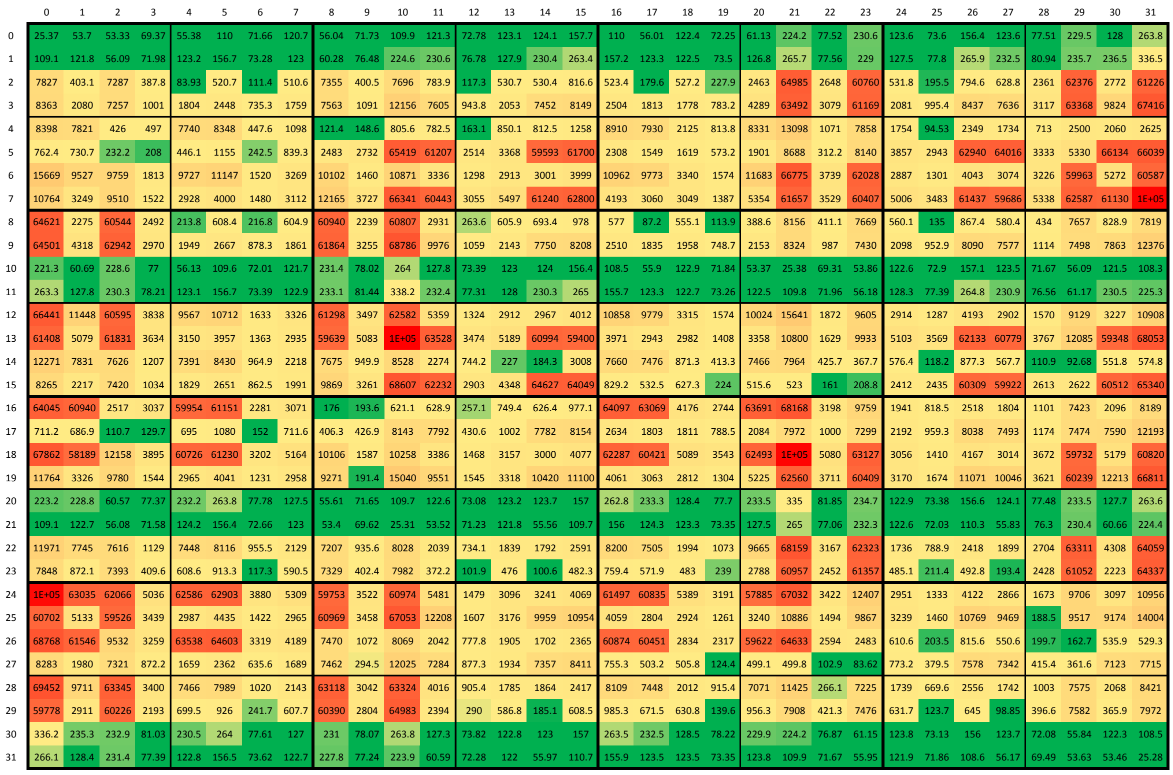
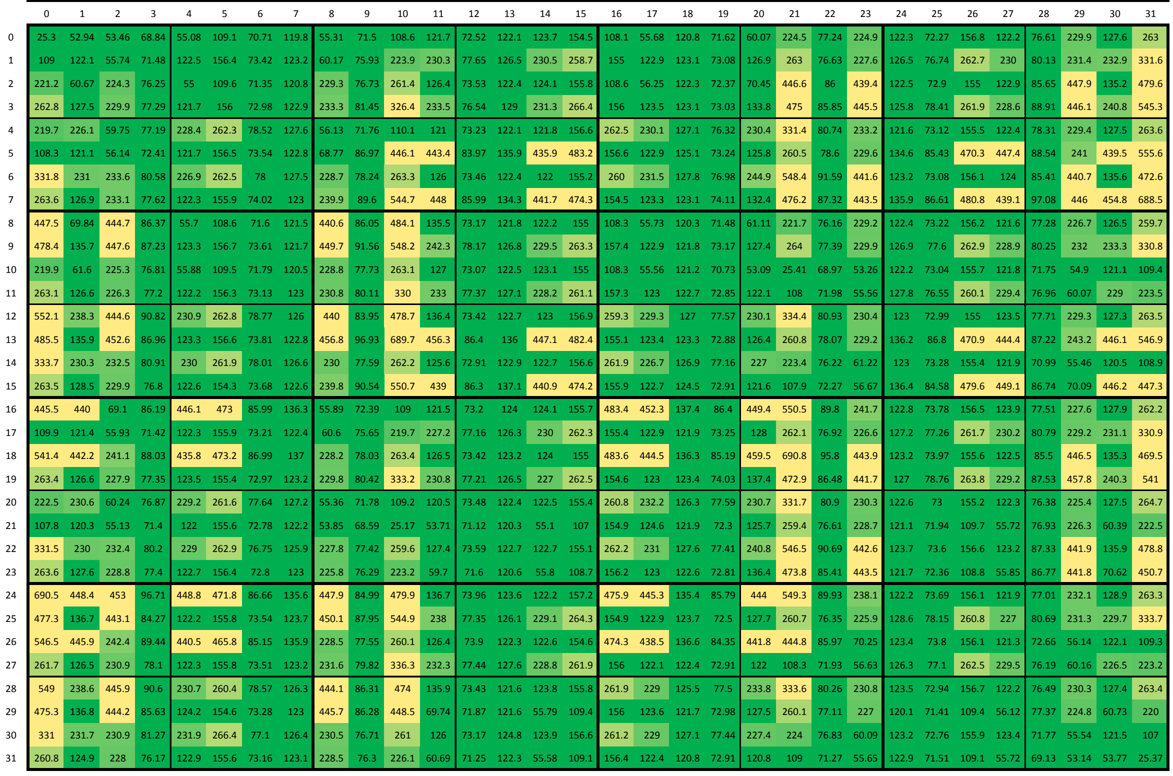


Figure 6.6: Frequency of latency in a mesh for some cores



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	25.37	53.7	53.33	69.37	55.38	110	71.66	120.7	56.04	71.73	109.9	121.3	72.78	123.1	124.1	157.7	110	56.01	122.4	72.25	61.13	224.2	77.52	230.6	123.6	73.6	156.4	123.6	77.51	229.5	128	263.8
1	109.1	121.8	56.09	71.98	123.2	156.7	73.28	123	60.28	76.48	224.6	230.6	76.78	127.9	230.4	263.4	157.2	123.3	122.5	73.5	126.8	265.7	77.56	229	127.5	77.8	265.9	232.5	80.94	235.7	236.5	336.5
2	7827	403.1	7287	387.8	83.93	520.7	111.4	510.6	7355	400.5	7696	783.9	117.3	530.7	530.4	816.6	523.4	179.6	527.2	227.9	2463	64985	2648	60760	531.8	195.5	794.6	628.8	2361	62376	2772	61226
3	8363	2080	7257	1001	1804	2448	735.3	1759	7563	1091	12156	7605	943.8	2053	7452	8149	2504	1813	1778	783.2	4289	63492	3079	61169	2081	995.4	8437	7636	3117	63368	9824	67416
4	8398	7821	426	497	7740	8348	447.6	1098	121.4	148.6	805.6	782.5	163.1	850.1	812.5	1258	8910	7930	2125	813.8	8331	13098	1071	7858	1754	94.53	2349	1734	713	2500	2060	2625
5	762.4	730.7	232.2	208	446.1	1155	242.5	839.3	2483	2732	65419	61207	2514	3368	59593	61700	2308	1549	1619	573.2	1901	8688	312.2	8140	3857	2943	62940	64016	3333	5330	66134	66039
6	15669	9527	9759	1813	9727	11147	1520	3269	10102	1460	10871	3336	1298	2913	3001	3999	10962	9773	3340	1574	11683	66775	3739	62028	2887	1301	4043	3074	3226	59963	5272	60587
7	10764	3249	9510	1522	2928	4000	1480	3112	12165	3727	66341	60443	3055	5497	61240	62800	4193	3060	3049	1387	5354	61657	3529	60407	5006	3483	61437	59686	5338	62587	61130	1E+05
8	64621	2275	60544	2492	213.8	608.4	216.8	604.9	60940	2239	60807	2931	263.6	605.9	693.4	978	577	87.2	555.1	113.9	388.6	8156	411.1	7669	560.1	135	867.4	580.4	434	7657	828.9	7819
9	64501	4318	62942	2970	1949	2667	878.3	1861	61864	3255	68786	9976	1059	2143	7750	8208	2510	1835	1958	748.7	2153	8324	987	7430	2098	952.9	8090	7577	1114	7498	7863	12376
10	221.3	60.69	228.6	77	56.13	109.6	72.01	121.7	231.4	78.02	264	127.8	73.39	123	124	156.4	108.5	55.9	122.9	71.84	53.37	25.38	69.31	53.86	122.6	72.9	157.1	123.5	71.67	56.09	121.5	108.3
11	263.3	127.8	230.3	78.21	123.1	156.7	73.39	122.9	233.1	81.44	338.2	232.4	77.31	128	230.3	265	155.7	123.3	122.7	73.26	122.5	109.8	71.96	56.18	128.3	77.39	264.8	230.9	76.56	61.17	230.5	225.3
12	66441	11448	60595	3838	9567	10712	1633	3326	61298	3497	62582	5359	1324	2912	2967	4012	10858	9779	3315	1574	10024	15641	1872	9605	2914	1287	4193	2902	1570	9129	3227	10908
13	61408	5079	61831	3634	3150	3957	1363	2935	59639	5083	1E+05	63528	3474	5189	60994	59400	3971	2943	2982	1408	3358	10800	1629	9933	5103	3569	62133	60779	3767	12085	59348	68053
14	12271	7831	7626	1207	7391	8430	964.9	2218	7675	949.9	8528	2274	744.2	227	184.3	3008	7660	7476	871.3	413.3	7466	7964	425.7	367.7	576.4	118.2	877.3	567.7	110.9	92.68	551.8	574.8
15	8265	2217	7420	1034	1829	2651	862.5	1991	9869	3261	68607	62232	2903	4348	64627	64049	829.2	532.5	627.3	224	515.6	523	161	208.8	2412	2435	60309	59922	2613	2622	60512	65340
16	64045	60940	2517	3037	59954	61151	2281	3071	176	193.6	621.1	628.9	257.1	749.4	626.4	977.1	64097	63069	4176	2744	63691	68168	3198	9759	1941	818.5	2518	1804	1101	7423	2096	8189
17	711.2	686.9	110.7	129.7	695	1080	152	711.6	406.3	426.9	8143	7792	430.6	1002	7782	8154	2634	1803	1811	788.5	2084	7972	1000	7299	2192	959.3	8038	7493	1174	7474	7590	12193
18	67862	58189	12158	3895	60726	61230	3202	5164	10106	1587	10258	3386	1468	3157	3000	4077	62287	60421	5089	3543	62493	1E+05	5080	63127	3056	1410	4167	3014	3672	59732	5179	60820
19	11764	3326	9780	1544	2965	4041	1231	2958	9271	191.4	15040	9551	1545	3318	10420	11100	4061	3063	2812	1304	5225	62560	3711	60409	3170	1674	11071	10046	3621	60239	12213	66811
20	223.2	228.8	60.57	77.37	232.2	263.8	77.78	127.5	55.61	71.65	109.7	122.6	73.08	123.2	123.7	157	262.8	233.3	128.4	77.7	233.5	335	81.85	234.7	122.9	73.38	156.6	124.1	77.48	233.5	127.7	263.6
21	109.1	122.7	56.08	71.58	124.2	156.4	72.66	123	53.4	69.62	25.31	53.52	71.23	121.8	55.56	109.7	156	124.3	123.3	73.35	127.5	265	77.06	232.3	122.6	72.03	110.3	55.83	76.3	230.4	60.66	224.4
22	11971	7745	7616	1129	7448	8116	955.5	2129	7207	935.6	8028	2039	734.1	1839	1792	2591	8200	7505	1994	1073	9665	68159	3167	62323	1736	788.9	2418	1899	2704	63311	4308	64059
23	7848	872.1	7393	409.6	608.6	913.3	117.3	590.5	7329	402.4	7982	372.2	101.9	476	100.6	482.3	759.4	571.9	483	239	2788	60957	2452	61357	485.1	211.4	492.8	193.4	2428	61052	2223	64337
24	1E+05	63035	62066	5036	62586	62903	3880	5309	59753	3522	60974	5481	1479	3096	3241	4069	61497	60835	5389	3191	57885	67032	3422	12407	2951	1333	4122	2866	1673	9706	3097	10956
25	60702	5133	59526	3439	2987	4435	1422	2965	60969	3458	67053	12208	1607	3176	9959	10954	4059	2804	2924	1261	3240	10886	1494	9867	3239	1460	10769	9469	188.5	9517	9174	14004
26	68768	61546	9532	3259	63538	64603	3319	4189	7470	1072	8069	2042	777.8	1905	1702	2365	60874	60451	2834	2317	59622	64633	2594	2483	610.6	203.5	815.6	550.6	199.7	162.7	535.9	529.3
27	8283	1980	7321	872.2	1659	2362	635.6	1689	7462	294.5	12025	7284	877.3	1934	7357	8411	755.3	503.2	505.8	124.4	499.1	499.8	102.9	83.62	773.2	379.5	7578	7342	415.4	361.6	7123	7715
28	69452	9711	63345	3400	7466	7989	1020	2143	63118	3042	63324	4016	905.4	1785	1864	2417	8109	7448	2012	915.4	7071	11425	266.1	7225	1739	669.6	2556	1742	1003	7575	2068	8421
29	59778	2911	60226	2193	699.5	926	241.7	607.7	60390	2804	64983	2394	290	586.8	185.1	608.5	985.3	671.5	630.8	139.6	956.3	7908	421.3	7476	631.7	123.7	645	98.85	396.6	7582	365.9	7972
30	336.2	235.3	232.9	81.03	230.5	264	77.61	127	231	78.07	263.8	127.3	73.82	122.8	123	157	263.5	232.5	128.5	78.22	229.9	224.2	76.87	61.15	123.8	73.13	156	123.7	72.08	55.84	122.3	108.5
31	266.1	128.4	231.4	77.39	122.8	156.5	73.62	122.7	227.8	77.24	223.9	60.59	72.28	122	55.97	110.7	155.9	123.5	123.5	73.35	123.8	109.9	71.67	55.95	121.9	71.86	108.6	56.17	69.49	53.63	53.46	25.28



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	25.3	52.94	53.46	68.84	55.08	109.1	70.71	119.8	55.31	71.5	108.6	121.7	72.52	122.1	123.7	154.5	108.1	55.68	120.8	71.62	60.07	224.5	77.24	224.9	122.3	72.27	156.8	122.2	76.61	229.9	127.6	263
1	109	122.1	55.74	71.48	122.5	156.4	73.42	123.2	60.17	75.93	223.9	230.3	77.65	126.5	230.5	258.7	155	122.9	123.1	73.08	126.9	263	76.63	227.6	126.5	76.74	262.7	230	80.13	231.4	232.9	331.6
2	221.2	60.67	224.3	76.25	55	109.6	71.35	120.8	229.3	76.73	261.4	126.4	73.53	122.4	124.1	155.8	108.6	56.25	122.3	72.37	70.45	446.6	86	439.4	122.5	72.9	155	122.9	85.65	447.9	135.2	479.6
3	262.8	127.5	229.9	77.29	121.7	156	72.98	122.9	233.3	81.45	326.4	233.5	76.54	129	231.3	266.4	156	123.5	123.1	73.03	133.8	475	85.85	445.5	125.8	78.41	261.9	228.6	88.91	446.1	240.8	545.3
4	219.7	226.1	59.75	77.19	228.4	262.3	78.52	127.6	56.13	71.76	110.1	121	73.23	122.1	121.8	156.6	262.5	230.1	127.1	76.32	230.4	331.4	80.74	233.2	121.6	73.12	155.5	122.4	78.31	229.4	127.5	263.6
5	108.3	121.1	56.14	72.41	121.7	156.5	73.54	122.8	68.77	86.97	446.1	443.4	83.97	135.9	435.9	483.2	156.6	122.9	125.1	73.24	125.8	260.5	78.6	229.6	134.6	85.43	470.3	447.4	88.54	241	439.5	555.6
6	331.8	231	233.6	80.58	226.9	262.5	78	127.5	228.7	78.24	263.3	126	73.46	122.4	122	155.2	260	231.5	127.8	76.98	244.9	548.4	91.59	441.6	123.2	73.08	156.1	124	85.41	440.7	135.6	472.6
7	263.6	126.9	233.1	77.62	122.3	155.9	74.02	123	239.9	89.6	544.7	448	85.99	134.3	441.7	474.3	154.5	123.3	123.1	74.11	132.4	476.2	87.32	443.5	135.9	86.61	480.8	439.1	97.08	446	454.8	688.5
8	447.5	69.84	444.7	86.37	55.7	108.6	71.6	121.5	440.6	86.05	484.1	135.5	73.17	121.8	122.2	155	108.3	55.73	120.3	71.48	61.11	221.7	76.16	229.2	122.4	73.22	156.2	121.6	77.28	226.7	126.5	259.7
9	478.4	135.7	447.6	87.23	123.3	156.7	73.61	121.7	449.7	91.56	548.2	242.3	78.17	126.8	229.5	263.3	157.4	122.9	121.8	73.17	127.4	264	77.39	229.9	126.9	77.6	262.9	228.9	80.25	232	233.3	330.8
10	219.9	61.6	225.3	76.81	55.88	109.5	71.79	120.5	228.8	77.73	263.1	127	73.07	122.5	123.1	155	108.3	55.56	121.2	70.73	53.09	25.41	68.97	53.26	122.2	73.04	155.7	121.8	71.75	54.9	121.1	109.4
11	263.1	126.6	226.3	77.2	122.2	156.3	73.13	123	230.8	80.11	330	233	77.37	127.1	228.2	261.1	157.3	123	122.7	72.85	122.1	108	71.98	55.56	127.8	76.55	260.1	229.4	76.96	60.07	229	223.5
12	552.1	238.3	444.6	90.82	230.9	262.8	78.77	126	440	83.95	478.7	136.4	73.42	122.7	123	156.9	259.3	229.3	127	77.57	230.1	334.4	80.93	230.4	123	72.99	155	123.5	77.71	229.3	127.3	263.5
13	485.5	135.9	452.6	86.96	123.3	156.6	73.81	122.8	456.8	96.93	689.7	456.3	86.4	136	447.1	482.4	155.1	123.4	123.3	72.88	126.4	260.8	78.07	229.2	136.2	86.8	470.9	444.4	87.22	243.2	446.1	546.9
14	333.7	230.3	232.5	80.91	230	261.9	78.01	126.6	230	77.59	262.2	125.6	72.91	122.9	122.7	156.6	261.9	226.7	126.9	77.16	227	223.4	76.22	61.22	123	73.28	155.4	121.9	70.99	55.46	120.5	108.9
15	263.5	128.5	229.9	76.8	122.6	154.3	73.68	122.6	239.8	90.54	550.7	439	86.3	137.1	440.9	474.2	155.9	122.7	124.5	72.91	121.6	107.9	72.27	56.67	136.4	84.58	479.6	449.1	86.74	70.09	446.2	447.3
16	445.5	440	69.1	86.19	446.1	473	85.99	136.3	55.89	72.39	109	121.5	73.2	124	124.1	155.7	483.4	452.3	137.4	86.4	449.4	550.5	89.8	241.7	122.8	73.78	156.5	123.9	77.51	227.6	127.9	262.2
17	109.9	121.4	55.93	71.42	122.3	155.9	73.21	122.4	60.6	75.65	219.7	227.2	77.16	126.3	230	262.3	155.4	122.9	121.9	73.25	128	262.1	76.92	226.6	127.2	77.26	261.7	230.2	80.79	229.2	231.1	330.9
18	541.4	442.2	241.1	88.03	435.8	473.2	86.99	137	228.2	78.03	263.4	126.5	73.42	123.2	124	155	483.6	444.5	136.3	85.19	459.5	690.8	95.8	443.9	123.2	73.97	155.6	122.5	85.5	446.5	135.3	469.5
19	263.4	126.6	227.9	77.35	123.5	155.4	72.97	123.2	229.8	80.42	333.2	230.8	77.21	126.5	227	262.5	154.6	123	123.4	74.03	137.4	472.9	86.48	441.7	127	78.76	263.8	229.2	87.53	457.8	240.3	541
20	222.5	230.6	60.24	76.87	229.2	261.6	77.64	127.2	55.36	71.78	109.2	120.5	73.48	122.4	122.5	155.4	260.8	232.2	126.3	77.59	230.7	331.7	80.9	230.3	122.6	73	155.2	122.3	76.38	225.4	127.5	264.7
21	107.8	120.3	55.13	71.4	122	155.6	72.78	122.2	53.85	68.59	25.17	53.71	71.12	120.3	55.1	107	154.9	124.6	121.9	72.3	125.7	259.4	76.61	228.7	121.1	71.94	109.7	55.72	76.93	226.3	60.39	222.5
22	331.5	230	232.4	80.2	229	262.9	76.75	125.9	227.8	77.42	259.6	127.4	73.59	122.7	122.7	155.1	262.2	231	127.6	77.41	240.8	546.5	90.69	442.6	123.7	73.6	156.6	123.2	87.33	441.9	135.9	478.8
23	263.6	127.6	228.8	77.4	122.7	156.4	72.8	123	225.8	76.29	223.2	59.7	71.6	120.6	55.8	108.7	156.2	123	122.6	72.81	136.4	473.8	85.41	443.5	121.7	72.36	108.8	55.85	86.77	441.8	70.62	450.7
24	690.5	448.4	453	96.71	448.8	471.8	86.66	135.6	447.9	84.99	479.9	136.7	73.96	123.6	122.2	157.2	475.9	445.3	135.4	85.79	444	549.3	89.93	238.1	122.2	73.69	156.1	121.9	77.01	232.1	128.9	263.3
25	477.3	136.7	443.1	84.27	122.2	155.8	73.54	123.7	450.1	87.95	544.9	238	77.35	126.1	229.1	264.3	154.9	122.9	123.7	72.5	127.7	260.7	76.35	225.9	128.6	78.15	260.8	227	80.69	231.3	229.7	333.7
26	546.5	445.9	242.4	89.44	440.5	465.8	85.15	135.9	228.5	77.55	260.1	126.4	73.9	122.3	122.6	154.6	474.3	438.5	136.6	84.35	441.8	444.8	85.97	70.25	123.4	73.8	156.1	121.3	72.66	56.14	122.1	109.3
27	261.7	126.5	230.9	78.1	122.3	155.8	73.51	123.2	231.6	79.82	336.3	232.3	77.44	127.6	228.8	261.9	156	122.1	122.4	72.91	122	108.3	71.93	56.63	126.3	77.1	262.5	229.5	76.19	60.16	226.5	223.2
28	549	238.6	445.9	90.6	230.7	260.4	78.57	126.3	444.1	86.31	474	135.9	73.43	121.6	123.8	155.8	261.9	229	125.5	77.5	233.8	333.6	80.26	230.8	123.5	72.94	156.7	122.2	76.49	230.3	127.4	263.4
29	475.3	136.8	444.2	85.63	124.2	154.6	73.28	123	445.7	86.28	448.5	69.74	71.87	121.6	55.79	109.4	156	123.6	121.7	72.98	127.5	260.1	77.11	227	120.1	71.41	109.4	56.12	77.37	224.8	60.73	220
30	331	231.7	230.9	81.27	231.9	266.4	77.1	126.4	230.5	76.71	261	126	73.17	124.8	123.9	156.6	261.2	229	127.1	77.44	227.4	224	76.83	60.09	123.2	72.76	155.9	123.4	71.77	55.54	121.5	107
31	260.8	124.9	228	76.17	122.9	155.6	73.16	123.1	228.5	76.3	226.1	60.69	71.25	122.3	55.58	109.1	156.4	122.4	120.8	72.91	120.8	109	71.27	55.65	122.9	71.51	109.1	55.72	69.13	53.14	53.77	25.37

Figure 6.7: Fat quadtree heatmap for ring clustering (the saturated case on top)

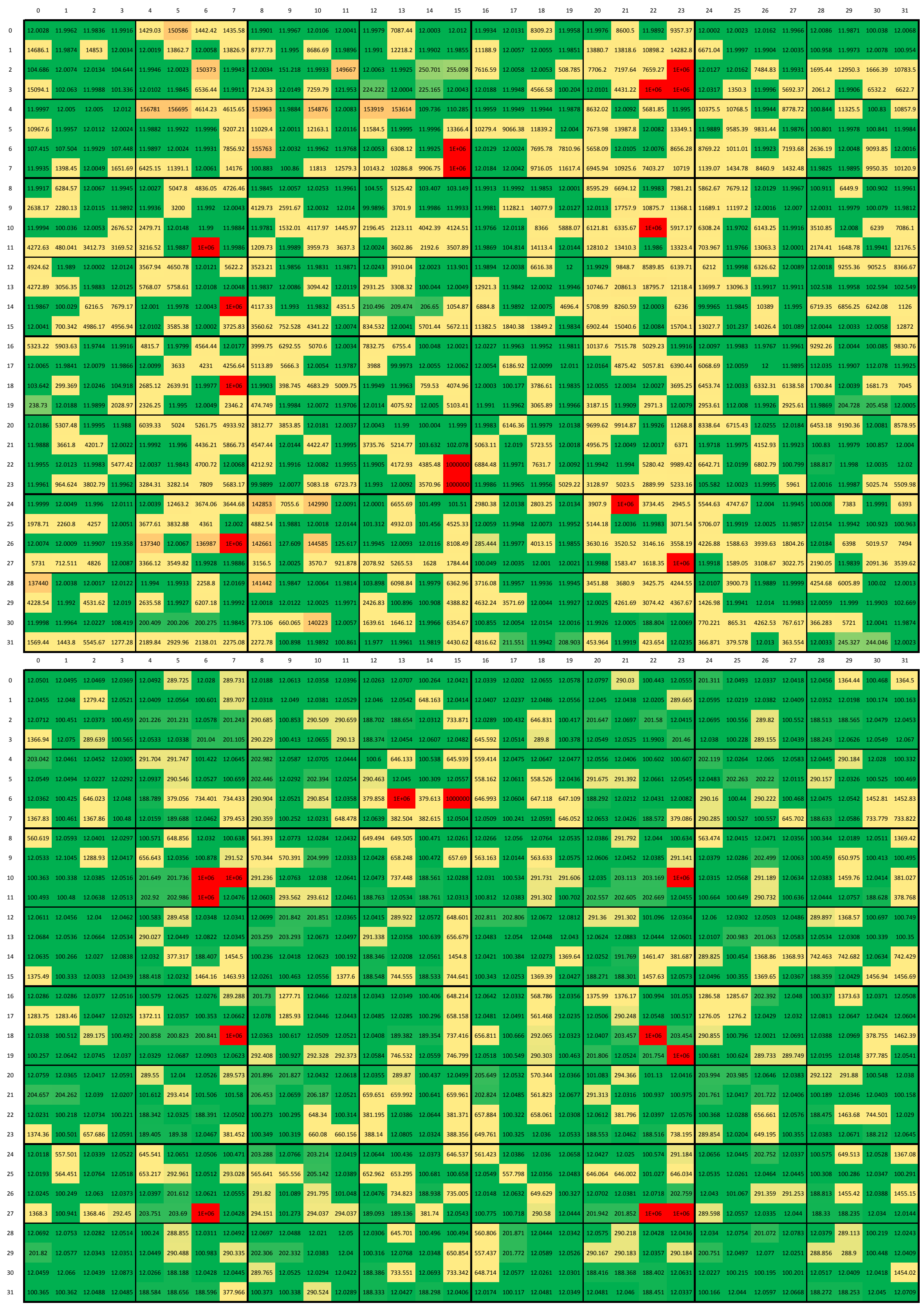


Figure 6.8: Cmesh heatmap for ring clustering (the saturated case on top)

6.5.2 Effect of Cache Coherency Traffic on Throughput

For group clustering, Figure 6.9, the fat quadtree and the concentrated mesh perform better than the mesh. However, as the write rate increases in the fat quadtree, the throughput also increases, then remains constant at nearly $2\,TB/s$, which is a worse case. The cmesh throughput increases as the memory access generation rate increases, and its highest memory bandwidth is around $7.5\,TB/s$.

For ring clustering, Figure 6.10, the concentrated mesh performs better than the fat quadtree and the mesh. Nevertheless, the fat quadtree throughput is marginally lower than the concentrated mesh. The highest memory bandwidth in the cmesh is around $4\,TB/s$. Overall, the throughput of the group clustering is higher than the throughput in the ring clustering.

Comparing the *streamcluster* application in our system in terms of throughput, we see that it has low throughput, and the system is not saturated with many memory access requests.

In today's systems, the PEZY-SC Many Core Processor has a memory bandwidth of $1.5\,TB/s$, taking into consideration that this chip is without cache coherency. Furthermore, Intel Xeon Phi has a memory bandwidth of $0.5\,TB/s$. Comparing the memory bandwidth of these systems with our system, we can see that our system performs better than today's systems in both group clustering and ring clustering. It is important to mention that the wire delay in our chip in 2023 is 6.7 times higher than the wire delay in chips today, as mentioned in Section 3.2.

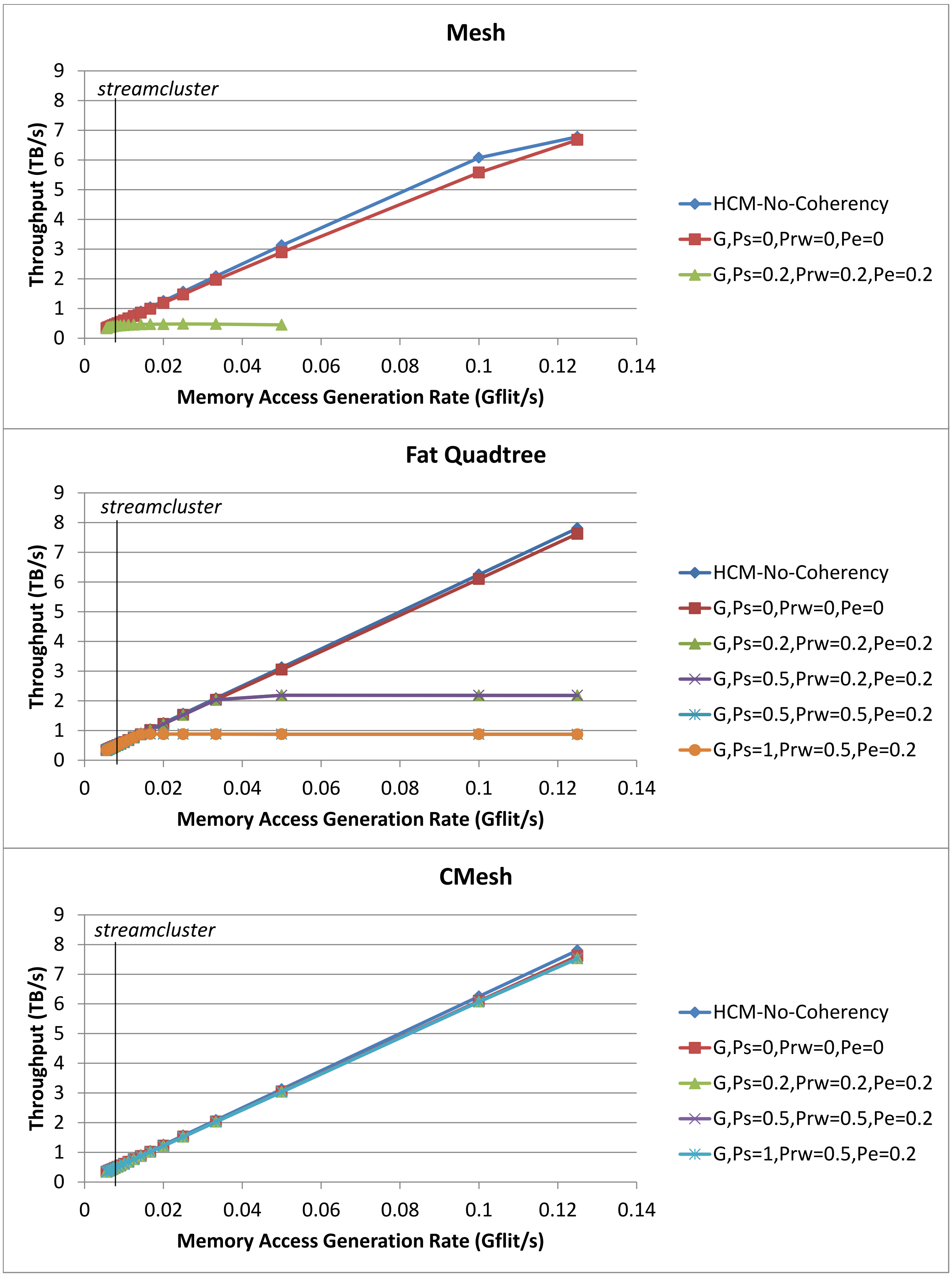


Figure 6.9: Throughput for group clustering (G: group clustering, P_s: the shared rate, P_{rw}: the read/write rate, and P_e: the eviction rate)

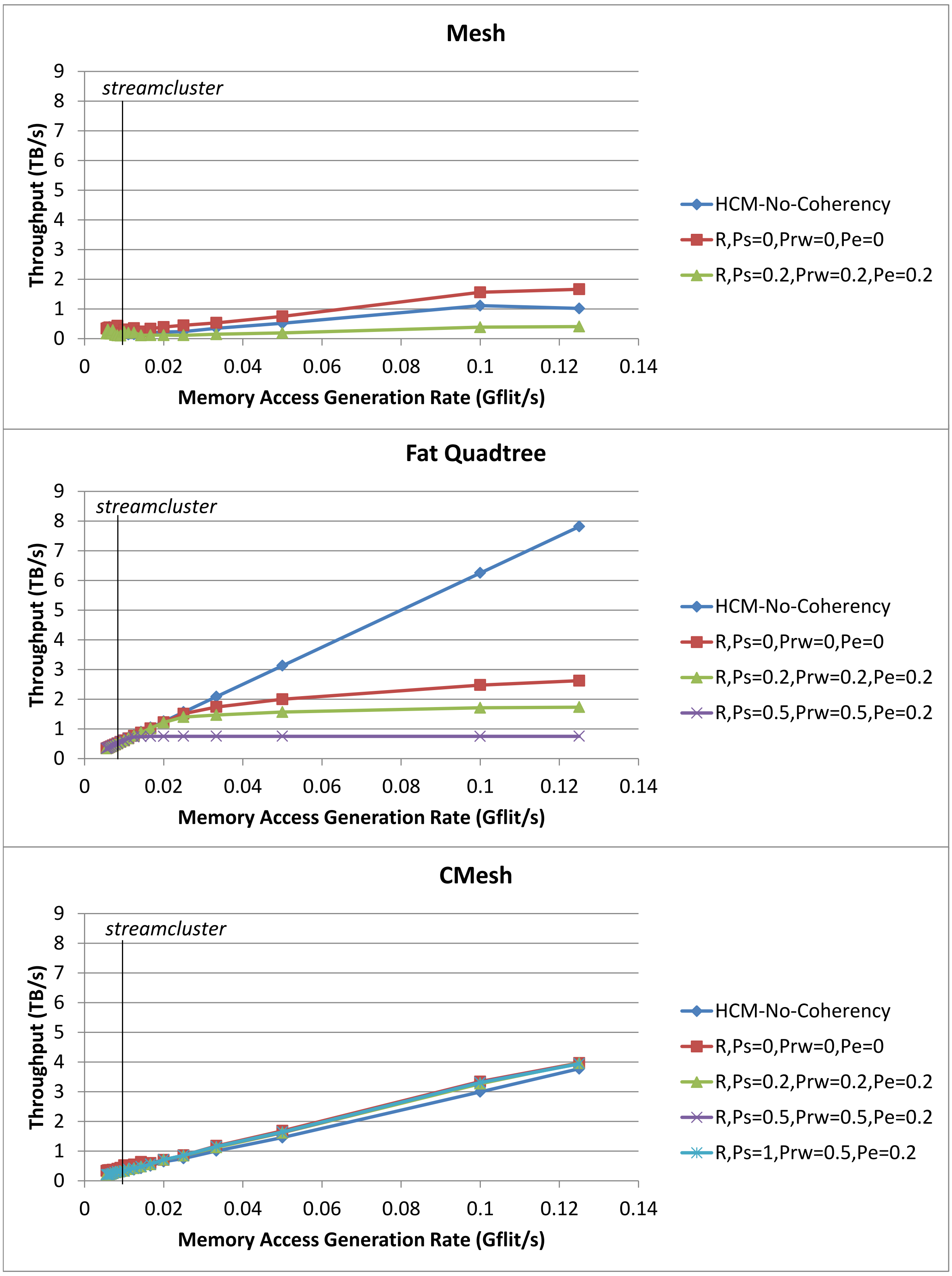


Figure 6.10: Throughput for ring clustering (R: ring clustering, P_s: the shared rate, P_{rw}: the read/write rate, and P_e: the eviction rate)

6.6 Summary

In this chapter, we suggested a directory-based cache coherency protocol. We have studied the cache coherency overhead and the performance of three different NoC topologies for future many-core systems. We used group clustering and ring clustering models for thread placement in shared memory systems. We showed that the directory-based protocol has only 0.97% overhead for each cache line in the cache. The concentrated mesh and the fat quadtree both perform better than the mesh. Our results clearly show that cache coherency scales and it does not affect the system greatly when using the hierarchical cache model. Furthermore, our results show that it is important to choose the right model for thread placement.

Chapter 7

Conclusion and Future Work

A general purpose processor used to consist of a single processing core. Due to the continuous increase of the number of transistors available on-chip, it became impossible to follow Moore's Law as a number of limitations occurred. This led to architectures with many independent processing cores being integrated into a single chip.

For these many-core architectures, a Network-on-Chip is required to interconnect the individual cores. The NoC is important because it increases the communication performance in comparison with single bus-based architectures.

Using 3D memory stacking on top of the many-core systems allows each processor core to have fast and high bandwidth access to the memory directly stacked on top of it using very dense vertical interconnects

From this we see that many-core systems are likely to continue in the future. Network-on-Chip is the dominant interconnection architecture. 3D memory stacking is the solution to breaking the memory wall. It has become clear from our literature review in Chapter 2 that the cache architectures and coherency protocols have not sufficiently advanced to provide scalability to thousands of cores. The goal of our research is to provide a scalable many-core architecture by exploiting locality in distributed memory architectures and shared memory architectures.

7.1 Thesis Statement Revisited

The thesis statement presented in Section 1.3 is as follows:

Memory architectures that support locality of computations on many-core Network-on-Chip (NoC) communication subsystems can improve the performance in terms of latency and throughput.

Distributed memory architecture and shared memory architecture can scale and improve their communication performance if locality of computations is introduced. Many-core systems that force cores to communicate mainly to their neighbouring cores can minimise the communication cost. Locality of computations can increase bandwidth and reduce latency. The ITRS physical data show that the wire delay will increase significantly in the future; therefore, the locality of computations will be crucial towards increasing many-core NoC performance.

To prove this statement, the following work has been done:

Chapter 3 described the cost models for the link complexity, routers, buffers, wire space, and area space for three topologies. It showed the technology node assumptions for realistic future many-core system parameters used in the research, based on data from the International Technology Roadmap for Semiconductors (ITRS) for the year 2023. Based on the cost models and the technology node assumptions, the chapter detailed the overheads for a 1024-core chip. It described packet format and switching techniques to improve the packet latency. The chapter discussed a number of NoC simulation tools and explained in detail the simulation tool selected for this work. A great deal of work has been done to HNOCS to suit the requirements of this research, as is shown in this chapter. HNOCS-XT, the extended version of HNOCS, is compared with Noxim to assess the accuracy, the running time, and the correctness of the simulator. The chapter described the experimental methodology proposed to exploit locality and two models of choosing neighbouring cores: group clustering and ring clustering. Finally, the chapter provided the performance measures used to evaluate the results of this work.

Chapter 4 explained the importance of locality of computation. It explained the relationship between our proposed hierarchical locality approach and Rent's rule for the bandwidth of the generated traffic. The chapter described the topologies used in this research: mesh, concentrated mesh, and fat quadtree. It described the proposed physical layout of the fat quadtree. In this chapter, we investigated the overhead and performance of flat (mesh, cmesh) and scale-invariant (fat quadtree) NoC topologies for future many-core systems with thousands of cores, under group clustering and ring clustering localisation models for point-to-point traffic. We have shown that the degree of locality and the clustering model strongly affect the performance of the network. Scale-invariant topologies such as the fat quadtree perform worse than flat ones (esp. the cmesh) because the reduced hop count is outweighed by the longer path delays, as a consequence of the high wire delay in the $10\,nm$ CMOS process. Our results clearly show the importance of traffic localisation for very large many-core systems.

Chapter 5 detailed the proposed hierarchical cache architecture model that benefits from communication locality. We suggested efficient hierarchical cache placements for caches on the mesh, the concentrated mesh, and the fat quadtree. The chapter described the cost models for links and buffers per virtual channel. In this chapter, we investigated the overhead and performance of flat (mesh, cmesh) and scale-invariant (fat quadtree) NoC topologies for future many-core systems with thousands of cores using two different models of localised thread placement in shared memory systems: group clustering and ring clustering. We showed that the distance between the threads sharing a memory block and the clustering model strongly affect the performance of the network. Scale-invariant topologies, such as the fat quadtree, perform better than flat ones because their structure matches the hierarchical cache architecture. In addition, the fat quadtree has a direct link between the levels of cache, although the wire delay increases as the request travels up the tree. Our results clearly showed the importance of localised thread placement for very large many-core systems.

Chapter 6 expressed the importance of cache coherency in shared memory architectures. It described the cache coherency protocol suggested, as well as showing the hierarchical tracking of sharers. In this chapter, we studied the cache coherency overhead and the performance of three different NoC topologies for future many-core systems. We used group clustering and ring clustering models for thread placement in shared memory systems. We showed that directory-based protocol has only 0.97% overhead for each cache line in the cache. The concentrated mesh and the fat quadtree both perform better than the mesh. Our results clearly showed that cache coherency scales, and it does not affect the system greatly when using the hierarchical cache model. Furthermore, our results showed that it is important to choose the right model for thread placement.

7.2 Summary of Research Contributions

This work contributed towards the scalability of many-core systems in the following findings:

- **The link complexity, routers, buffers, area space, and wire space overhead of the NoC on a thousand-core system is negligible, and the choice of topology depends solely on the performance.**
  In order to test the research hypothesis, the technology node assumptions were made on data from the International Technology Roadmap for Semiconductors (ITRS) for the year 2023. The cost model for a number of topologies was calculated to show that the overhead of the NoC on a thousand-core system is negligible.

- **In distributed memory architecture, locality of computations on many-core NoC communication subsystems improves the system performance in terms of latency and throughput. Scale-invariant topologies, such as the fat quadtree, perform worse than flat ones (esp. cmesh).**
  We showed that the degree of locality and the clustering model strongly affect the

performance of the network. Scale-invariant topologies, such as the fat quadtree, performed worse than flat ones (esp. cmesh) because the reduced hop count is outweighed by the longer path delays, as a consequence of the high wire delay in the $10\,nm$ CMOS process. Our results clearly showed the importance of traffic localisation for very large many-core systems.

- **In shared memory architecture, data locality on 3D stacked hierarchical cache architecture for many-core NoC communication subsystems without cache coherency traffic improves the system performance in terms of latency and throughput. Scale-invariant topologies, such as the fat quadtree, perform better than flat ones.**
  We proposed a hierarchical cache model and suggested the cache placements in the mesh, the concentrated mesh, and the fat quadtree. We investigated the overhead and performance of flat (mesh, cmesh) and scale-invariant (fat quadtree) NoC topologies for future many-core systems with thousands of cores using two different models of localised thread placement in shared memory systems: group clustering and ring clustering. We showed that the distance between the threads sharing a memory block and the clustering model strongly affect the performance of the network. Scale-invariant topologies, such as the fat quadtree, performed better than flat ones because their structure matches the hierarchical cache architecture. In addition, the fat quadtree has a direct link between the levels of cache, although the wire delay increases as the requests travel up the tree. Our results clearly showed the importance of data locality and thread placement for very large many-core systems.

- **Cache coherency scales using hierarchical cache architecture and locality of computations with only slight overhead.**
  We explained a directory-based cache coherency protocol and a hierarchical method for tracking sharers. We have studied the cache coherency overhead and the performance of three different NoC topologies for future many-core systems. We used group clustering and ring clustering models for thread placement in shared memory architecture. We showed that directory-based protocol had only 0.97% overhead for each cache line in the cache. The concentrated mesh and the fat quadtree both performed better than the mesh. Our results clearly showed that cache coherency scales and it does not affect the system greatly when using the hierarchical cache model. Furthermore, our results showed that it is important to choose the right model for thread placement.

7.3 Future Directions of the Research

The work described in this dissertation has answered the questions posed by the research hypothesis, and in doing so has laid the foundation for significant future work. This section outlines a number of directions to be followed.

- 3D NoC topologies:
  The 2D integrated circuits have restricted floor planning choices and they limit the performance enhancements of NoC architectures. 3D integrated circuits allow for performance improvements because of the reduction in interconnection length. Besides this benefit, power is reduced from shorter wires.

- Locality aware scheduling on operating system level:
  If the operating system is aware of the hierarchical memory architecture that supports locality in the hardware level, it can take advantage of that by scheduling tasks in a way that supports locality of computations. An operating system that guarantees locality can improve the many-core performance even further.

- Locality aware programming models:
  Programming models that support locality can allow the programmer to take advantage of the hierarchical memory architecture which supports locality, hence increasing the many-core performance. This can be done by adding annotations in existing languages similar to the Java annotations [106].

- Cache coherency:
  Cache coherency protocols can be improved so that they generate less coherency traffic, which improves the many-core performance. Reducing the coherency traffic can be achieved by running threads that share a memory address in the same core or in cores close to each other.

Appendix A

A.1 Publications

A selection of the work presented in this thesis has been peer-reviewed and published in academic conference proceedings as follows:

- "The Impact of Traffic Localisation on the Performance of NoCs for Very Large Manycore Systems."
  S. Al Khanjari and W. Vanderbauwhede
  Procedia Computer Science 56 (2015): 403-408.
- "Evaluation of the Memory Communication Traffic in a Hierarchical Cache Model for Massively-Manycore Processors."
  S. Al Khanjari and W. Vanderbauwhede
  2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). IEEE, 2016.
- "The Performance of NoCs for Very Large Manycore Systems under Locality-based Traffic."
  S. Al Khanjari and W. Vanderbauwhede
  International Journal of Computing and Digital Systems. Volume : 5. Issue: 2. Issue Publication Date: March 2016.

A collaborative work on topics related to the main topics of the thesis resulted in the following publications:

- "Shortest Path Routing Algorithm for Hierarchical Interconnection Network-on-Chip."
  O. Inam, S. Al Khanjari, and W. Vanderbauwhede
  Procedia Computer Science 56 (2015): 409-414.
- "Group based Shortest Path Routing Algorithm for Hierarchical Cross Connected Recursive Networks (HCCR)."

O. Inam, S. Al Khanjari, and W. Vanderbauwhede
International Journal of Computing and Digital Systems. Volume : 5. Issue: 2. Issue Publication Date: March 2016.

We intend to publish the outcomes of Chapter 6, which is about cache coherency scales using hierarchical cache architecture and locality of computations with only slight overhead, in the near future.

Bibliography

[1] J. Jung, K. Kang, G. De Micheli, and C.-M. Kyung, "Runtime 3-d stacked cache management for chip-multiprocessors," 2013.

[2] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International.* IEEE, 2008, pp. 88–598.

[3] A. Duran and M. Klemm, "The intel® many integrated core architecture," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on.* IEEE, 2012, pp. 365–366.

[4] M. Moadeli, A. Shahrabi, W. Vanderbauwhede, and M. Ould-Khaoua, "An analytical performance model for the Spidergon NoC," *21st IEEE International Conference on Advanced Information Networking and Applications*, pp. 1014–1021, 2007.

[5] M. Moadeli, P. Maji, and W. Vanderbauwhede, "Quarc: A high-efficiency network on-chip architecture," in *Advanced Information Networking and Applications, 2009. AINA'09. International Conference on.* IEEE, Conference Proceedings, pp. 98–105.

[6] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *Proceedings of the 20th annual international conference on Supercomputing.* ACM, 2006, pp. 187–198.

[7] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *Computers, IEEE Transactions on*, vol. 100, no. 10, pp. 892–901, 1985.

[8] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 453–464.

[9] S. K. Lim, *3D-MAPS: 3D massively parallel processor with stacked memory.* Springer, 2013, pp. 537–560.

[10] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso *et al.*, “Die stacking (3d) microarchitecture,” in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’06).* IEEE, 2006, pp. 469–479.

[11] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, “Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement,” in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE.* IEEE, 2008, pp. 554–559.

[12] G. Russell, “Intel micron hybrid memory cube: The future of exascale computing,” *Bright Side of News. Sep*, vol. 19, 2011.

[13] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, “Optimizing communication and capacity in a 3d stacked reconfigurable cache hierarchy,” in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on.* IEEE, Conference Proceedings, pp. 262–274.

[14] A. Zia, P. Jacob, J.-W. Kim, M. Chu, R. P. Kraft, and J. F. McDonald, “A 3-d cache with ultra-wide data bus for 3-d processor-memory integration,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 967–977, 2010.

[15] Y. Xie, “Processor architecture design using 3d integration technology,” in *2010 23rd International Conference on VLSI Design.* IEEE, 2010, pp. 446–451.

[16] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny, “Hnocs: Modular open-source simulator for heterogeneous nocs,” in *Embedded Computer Systems (SAMOS), 2012 International Conference on.* IEEE, 2012, pp. 51–57.

[17] T. Bjerregaard and S. Mahadevan, “A survey of research and practices of network-on-chip,” *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 1, 2006.

[18] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, and G. Schrom, “A 48-core ia-32 message-passing processor with dvfs in 45nm cmos,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International.* IEEE, Conference Proceedings, pp. 108–109.

[19] C. A. Zeferino, M. E. Kreutz, and A. A. Susin, “Rasoc: A router soft-core for networks-on-chip,” in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3. IEEE, Conference Proceedings, pp. 198–203.

[20] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection networks*. Morgan Kaufmann, 2003.

[21] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*. IEEE, Conference Proceedings, pp. 684–689.

[22] Z. Xiao and B. Baas, "A hexagonal shaped processor and interconnect topology for tightly-tiled many-core architecture," in *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*. IEEE, Conference Proceedings, pp. 153–158.

[23] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," in *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*. IEEE, Conference Proceedings, p. 15.

[24] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Conference Proceedings, pp. 172–182.

[25] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *Computers, IEEE Transactions on*, vol. 54, no. 8, pp. 1025–1040, 2005.

[26] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries *et al.*, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 173–183, 2011.

[27] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob *et al.*, "An 80-tile 1.28 tflops network-on-chip in 65nm cmos," in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*. IEEE, 2007, pp. 98–589.

[28] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu, "Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip," in *2009 WRI Global Congress on Intelligent Systems*, vol. 3. IEEE, 2009, pp. 329–333.

[29] J. Camacho, J. Flich, A. Roca, and J. Duato, "Pc-mesh: A dynamic parallel concentrated mesh," in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 642–651.

[30] A. Kamath, G. Saxena, and B. Talawar, “Analysis of ring topology for noc architecture,” in *2015 International Conference on Computing and Network Communications (CoCoNet).* IEEE, 2015, pp. 381–388.

[31] D. C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, H. P. Hofstee, C. Johns *et al.*, “Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 179–196, 2006.

[32] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin *et al.*, “Larrabee: a many-core x86 architecture for visual computing,” in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 18.

[33] J. Reinders, “An overview of programming for intel xeon processors and intel xeon phi coprocessors,” *Intel Corporation, Santa Clara*, 2012.

[34] E. Salminen, A. Kulmala, and T. D. Hamalainen, “Survey of network-on-chip proposals,” *white paper, OCP-IP*, pp. 1–13, 2008.

[35] A. Agarwal, C. Iskander, and R. Shankar, “Survey of network on chip (noc) architectures & contributions,” *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[36] S. A. Przybylski, *Cache and memory hierarchy design: a performance-directed approach.* Morgan Kaufmann, 1990.

[37] G. Sun, *Exploring Memory Hierarchy Design with Emerging Memory Technologies.* Springer, 2014.

[38] A. S. Tanenbaum, *Structured computer organization.* Pearson, 2006.

[39] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, “Power7: Ibm’s next-generation server processor,” *IEEE micro*, vol. 30, no. 2, pp. 7–15, 2010.

[40] M. Butler, “Amd bulldozer core-a new approach to multithreaded compute performance for maximum efficiency and throughput,” in *IEEE HotChips Symposium on High-Performance Chips (HotChips 2010)*, 2010.

[41] F. Busaba, M. A. Blake, B. Curran, M. Fee, C. Jacobi, P.-K. Mak, B. R. Prasky, and C. R. Walters, “Ibm zenterprise 196 microprocessor and cache subsystem,” *IBM Journal of Research and Development*, vol. 56, no. 1.2, pp. 1–1, 2012.

[42] H. F. Jordan, “Shared versus distributed memory multiprocessors,” DTIC Document, Tech. Rep., 1991.

[43] H. Kasim, V. March, R. Zhang, and S. See, “Survey on parallel programming model,” in *IFIP International Conference on Network and Parallel Computing*. Springer, 2008, pp. 266–275.

[44] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel computer architecture: a hardware/-software approach*. Gulf Professional Publishing, 1999.

[45] N. Manchanda and K. Anand, “Non-uniform memory access (numa),” *New York University*, 2010.

[46] J. Lira, C. Molina, and A. Gonzlez, “Analysis of non-uniform cache architecture policies for chip-multiprocessors using the parsec benchmark suite,” in *Workshop on Managed Many-Core Systems (MMCS)*, Conference Proceedings.

[47] C. Kim, D. Burger, and S. W. Keckler, “An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches,” in *Acm Sigplan Notices*, vol. 37. ACM, Conference Proceedings, pp. 211–222.

[48] ——, “Nonuniform cache architectures for wire-delay dominated on-chip caches,” *Micro, IEEE*, vol. 23, no. 6, pp. 99–107, 2003.

[49] C. Lee, “Distributed shared memory,” in *Proceedings on the 15th CISL Winter Workshop Kushu, Japan, February*. Citeseer, 2002.

[50] B. Choi, R. Komuravelli, H. Sung, R. Bocchino, S. Adve, and V. Adve, “Denovo: Rethinking hardware for disciplined parallelism,” in *Proceedings of the Second USENIX Workshop on Hot Topics in Parallelism (HotPar)*, Conference Proceedings.

[51] J. H. Kelm, D. R. Johnson, W. Tuohy, S. S. Lumetta, and S. J. Patel, “Cohesion: An adaptive hybrid memory model for accelerators,” *Micro, IEEE*, vol. 31, no. 1, pp. 42–55, 2011.

[52] M. M. Martin, M. D. Hill, and D. J. Sorin, “Why on-chip cache coherence is here to stay,” *Communications of the ACM*, vol. 55, no. 7, pp. 78–89, 2012.

[53] S. J. Eggers and R. H. Katz, *Evaluating the performance of four snooping cache coherency protocols*. ACM, 1989, vol. 17, no. 3.

[54] M. Dalui and B. K. Sikdar, “An efficient test design for cmps cache coherence realizing mesi protocol,” in *Progress in VLSI Design and Test*. Springer, 2012, pp. 89–98.

[55] K. Baukus and R. Van Der Meyden, "A knowledge based analysis of cache coherence," in *International Conference on Formal Engineering Methods*. Springer, 2004, pp. 99–114.

[56] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *Computer*, vol. 23, no. 6, pp. 49–58, 1990.

[57] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, *The directory-based cache coherence protocol for the DASH multiprocessor*. ACM, 1990, vol. 18, no. 2SI.

[58] J. Laudon and D. Lenoski, "The sgi origin: a ccnuma highly scalable server," in *ACM SIGARCH Computer Architecture News*, vol. 25, no. 2. ACM, 1997, pp. 241–251.

[59] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, C. Watnik, A. T. Tran, and Z. Xiao, "A 167-processor computational platform in 65 nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 4, pp. 1130–1144, 2009.

[60] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Kilo-noc: a heterogeneous network-on-chip architecture for scalability and service guarantees," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3. ACM, 2011, Conference Proceedings, pp. 401–412.

[61] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, Conference Proceedings, pp. 746–749.

[62] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. G. Shet, G. Chrysos, and P. Dubey, "Design and implementation of the linpack benchmark for single and multi-node systems based on intel® xeon phi coprocessor," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 126–137.

[63] C. Demerjian, "Intel details knights corner architecture at long last," August 2016. [Online]. Available: http://semiaccurate.com/2012/08/28/intel-details-knights-corner-architecture-at-long-last/

[64] M. Berezecki, E. Frachtenberg, M. Paleczny, and K. Steele, "Power and performance evaluation of memcached on the tilepro64 architecture," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 2, pp. 81–90, 2012.

[65] ——, “Many-core key-value store,” in *Green Computing Conference and Workshops (IGCC), 2011 International*. IEEE, 2011, pp. 1–8.

[66] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao *et al.*, “The sunway taihulight supercomputer: system and applications,” *Science China Information Sciences*, vol. 59, no. 7, p. 072001, 2016.

[67] T. Aoyama, K.-I. Ishikawa, Y. Kimura, H. Matsufuru, A. Sato, T. Suzuki, and S. Torii, “First application of lattice qcd to pezy-sc processor,” *Procedia Computer Science*, vol. 80, pp. 1418–1427, 2016.

[68] T. Ishikawa, T. Boku, and M. Sato, “Design and preliminary evaluation of omni openacc compiler for massive mimd processor pezy-sc,” *OpenMP: Memory, Devices, and Tasks*, p. 293.

[69] S. Kundu and S. Chattopadhyay, *Network-on-chip: The Next Generation of System-on-chip Integration*. CRC Press, 2014.

[70] (2011) International technology roadmap for semiconductors (itrs).

[71] C. Killebrew *et al.*, “L2 cache to off-chip memory networks for chip multiprocessor,” Ph.D. dissertation, Department of Electrical Engineering and Computer Sciences, University of California, 2008.

[72] J. Fang, A. L. Varbanescu, H. Sips, L. Zhang, Y. Che, and C. Xu, “An empirical study of intel xeon phi,” *arXiv preprint arXiv:1310.5842*, 2013.

[73] J. Lee, C. Nicopoulos, S. J. Park, M. Swaminathan, and J. Kim, “Do we need wide flits in networks-on-chip?” in *VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on*. IEEE, 2013, pp. 2–7.

[74] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan, “Nirgam: a simulator for noc interconnect routing and application modeling,” in *Design, Automation and Test in Europe Conference*, 2007.

[75] M. Bauer and N. Jiang, “Parallelizing a network simulator,” *Network*, vol. 2, p. R3.

[76] F. Fazzino, M. Palesi, and D. Patti, “Noxim: Network-on-chip simulator,” *URL: http://sourceforge. net/projects/noxim*, 2008.

[77] A. Varga *et al.*, “The omnet++ discrete event simulation system,” in *Proceedings of the European Simulation Multiconference (ESM2001)*, vol. 9. sn, 2001, p. 185.

[78] A. Butko, R. Garibotti, L. Ost, V. Lapotre, A. Gamatie, G. Sassatelli, and C. Adeniyi-Jones, “A trace-driven approach for fast and accurate simulation of manycore architectures,” in *The 20th Asia and South Pacific Design Automation Conference*. IEEE, 2015, pp. 707–712.

[79] M. Badr and N. E. Jerger, “Synfull: synthetic traffic models capturing cache coherent behaviour,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 109–120.

[80] S. Nürnberger, G. Drescher, R. Rotta, J. Nolte, and W. Schröder-Preikschat, “Shared memory in the many-core age,” in *European Conference on Parallel Processing*. Springer, 2014, pp. 351–362.

[81] J. M. Smith and D. J. Farber, “Traffic characteristics of a distributed memory system,” *Computer Networks and ISDN Systems*, vol. 22, no. 2, pp. 143–154, 1991.

[82] A. Bhatele, H. Langer, A. D. Malony, M. Schulz, S. Shende, and N. R. Tallent, “Performance analysis, modeling and scaling of hpc applications and tools,” Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2016.

[83] R. Das, S. Eachempati, A. K. Mishra, V. Narayanan, and C. R. Das, “Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps,” in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*. IEEE, 2009, pp. 175–186.

[84] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, “Effect of traffic localization on energy dissipation in noc-based interconnect,” in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*. IEEE, 2005, pp. 1774–1777.

[85] B. M. Beckmann and D. A. Wood, “Managing wire delay in large chip-multiprocessor caches,” in *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*. IEEE, 2004, pp. 319–330.

[86] Z. Chishti, M. D. Powell, and T. Vijaykumar, “Optimizing replication, communication, and capacity allocation in cmps,” in *32nd International Symposium on Computer Architecture (ISCA’05)*. IEEE, 2005, pp. 357–368.

[87] D. Greenfield, A. Banerjee, J.-G. Lee, and S. Moore, “Implications of rent’s rule for noc design and its fault-tolerance,” in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*. IEEE, 2007, pp. 283–294.

[88] B. S. Landman and R. L. Russo, “On a pin versus block relationship for partitions of logic graphs,” *Computers, IEEE Transactions on*, vol. 100, no. 12, pp. 1469–1479, 1971.

[89] W. Heirman, J. Dambre, D. Stroobandt, and J. Van Campenhout, “Rent’s rule and parallel programs: characterizing network traffic behavior,” in *Proceedings of the 2008 international workshop on System level interconnect prediction*. ACM, 2008, pp. 87–94.

[90] M. Tripathy and C. Tripathy, “A comparative analysis of performance of shared memory cluster computing interconnection systems,” *Journal of Computer Networks and Communications*, vol. 2014, 2014.

[91] E. Guthmuller, I. Miro-Panades, and A. Greiner, “Adaptive stackable 3d cache architecture for manycores,” in *2012 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2012, pp. 39–44.

[92] S. K. Lim, “3d-maps: 3d massively parallel processor with stacked memory,” in *Design for High Performance, Low Power, and Reliable 3D Integrated Circuits*. Springer, 2013, pp. 537–560.

[93] S. Lee, K. Kang, J. Jung, and C.-M. Kyung, “Runtime 3-d stacked cache data management for energy minimization of 3-d chip-multiprocessors,” in *Fifteenth International Symposium on Quality Electronic Design*. IEEE, 2014, pp. 197–204.

[94] F. Broquedis, N. Furmento, B. Goglin, R. Namyst, and P.-A. Wacrenier, “Dynamic task and data placement over numa architectures: an openmp runtime perspective,” in *Evolving OpenMP in an Age of Extreme Parallelism*. Springer, 2009, pp. 79–92.

[95] D. Tam, R. Azimi, and M. Stumm, “Thread clustering: sharing-aware scheduling on smp-cmp-smt multiprocessors,” in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 47–58.

[96] E. Z. Zhang, Y. Jiang, and X. Shen, “Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs?” in *ACM Sigplan Notices*, vol. 45, no. 5. ACM, 2010, pp. 203–212.

[97] S. Mittal and J. Vetter, “A survey of techniques for architecting dram caches,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[98] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, “A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies,” in *Computer Architecture, 2008. ISCA’08. 35th International Symposium on*. IEEE, 2008, pp. 51–62.

[99] C. M. Krishna, *Performance modeling for computer architects*. John Wiley & Sons, 1996, vol. 11.

[100] G. Girão, B. C. de Oliveira, R. Soares, and I. S. Silva, "Cache coherency communication cost in a noc-based mpsoc platform," in *Proceedings of the 20th annual conference on Integrated circuits and systems design*. ACM, 2007, pp. 288–293.

[101] L. Han, J. An, D. Gao, X. Fan, X. Ren, and T. Yao, "A survey on cache coherence for tiled many-core processor," in *Signal Processing, Communication and Computing (ICSPCC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 114–118.

[102] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 169–180.

[103] H. Lee, S. Cho, and B. R. Childers, "Perfectory: a fault-tolerant directory memory architecture," *IEEE Transactions on Computers*, vol. 59, no. 5, pp. 638–650, 2010.

[104] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *ACM SIGARCH Computer Architecture News*, vol. 23, no. 2. ACM, 1995, pp. 24–36.

[105] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[106] R. McIlroy, "Using program behaviour to exploit heterogeneous multi-core processors," Ph.D. dissertation, Citeseer, 2010.