

ENABLING INTERPOSER-BASED DISINTEGRATION OF MULTI-CORE PROCESSORS

by

Ajaykumar Kannan

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
in the The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto

© Copyright 2015 by Ajaykumar Kannan

Abstract

Enabling Interposer-based Disintegration of Multi-core Processors

Ajaykumar Kannan

Master of Applied Science

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering

University of Toronto

2015

Silicon interposers enable high-performance processors to integrate a significant amount of in-package memory, thereby providing huge bandwidth gains while reducing the costs of accessing memory. Once the price has been paid for the interposer, there are new opportunities to exploit it and provide other system benefits. We consider how the routing resources afforded by the interposer can be used to improve the network-on-chip's (NoC) capabilities and use the interposer to "disintegrate" a multi-core chip into smaller chips that individually and collectively cost less to manufacture than a large monolithic chip. However, distributing a system across many pieces of silicon causes the overall NoC to become fragmented, thereby decreasing performance as core-to-core communications between different chips must now be routed through the interposer. We study the performance-cost trade-offs of implementing an interposer-based, multi-chip, multi-core system and propose new interposer NoC organizations to mitigate the performance challenges while preserving the cost benefits.

Acknowledgements

First, I would like to express my sincerest gratitude to my supervisor, Natalie Enright Jerger, for her guidance and motivation during my time here. I have learned a lot from her expertise in computer architecture, on-chip networks, and research methodologies. I have been very lucky in having her as my mentor and she has been just wonderful to work with.

I would also like to thank Gabriel Loh at AMD Corp., for his numerous inputs, suggestions and guidance throughout the work that I have been a part of during my programme. It has been a great opportunity and learning experience to have worked with him.

I also extend my thanks to the NEJ research group for all the support, help, and guidance. It has been a privilege to have had the chance to interact and discuss various ideas and subjects with them, technical and otherwise, and get their invaluable feedback. I would also like to thank the graduate students in Prof. Andreas Moshovos' research group for the help during those crucial times, and their feedback on our work.

I would like to thank my committee- professors Andreas Moshovos, Jason Anderson, and Josh Taylor for their insights and feedback on our work.

I would like to thank my friends and family who have helped me along the way, knowingly or unknowingly. Finally, I would like to thank my parents who started me on this path a long time ago and helped me get to this point in my life. I would not be here without them.

Contents

1	Introduction	1
1.1	Research Highlights	2
1.2	Organization	2
2	Background	3
2.1	Network on Chip	4
2.1.1	Network Parameters and Metrics	4
2.1.2	Network Topology	5
2.2	Die-Stacking	6
2.2.1	2.5D vs. 3D Stacking	7
2.3	Silicon Interposers and Their Networks	8
3	Motivation	9
3.1	Chip Disintegration	9
3.1.1	Chip Cost Analysis	10
3.2	Integration of smaller chips	12
3.2.1	Silicon Interposer-Based Chip Integration	13
3.2.2	Limitations of Cost Analysis	15
3.3	The Research Problem	15
4	NoC Architecture	17
4.1	Baseline Architecture	17
4.2	Routing Protocol	17
4.3	Network Topology	19
4.3.1	Misaligned Topologies	20
4.3.2	The ButterDonut Topology	22
4.3.3	Comparing Topologies	23
4.3.4	Deadlock Freedom	24
4.4	Physical implementation	24
4.4.1	μ bump Overheads	25
5	Methodology and Evaluation	27
5.1	Methodology	27
5.1.1	Synthetic Workloads	27
5.1.2	SynFull Simulations	28
5.1.3	Full-System Simulations	29
5.2	Experimental Evaluation	30
5.2.1	Performance	30
5.2.2	Load vs. Latency Analysis	33
5.2.3	Routing Protocols	35
5.2.4	Power and Area Modelling	35
5.2.5	Combining Cost and Performance	37
5.2.6	Clocking Across Chips	38

5.2.7	Non-disintegrated Interposer NoCs	39
6	Related Work	41
7	Future Directions & Conclusions	43
7.1	Future Directions	43
7.1.1	New Chip-design Concerns	43
7.1.2	Software-based Mitigation of Multi-Chip Interposer Effects	43
7.2	Conclusion	44
	Bibliography	45

List of Tables

3.1	Yield Analysis for multi-core chips	11
3.2	Parameters for the chip yield calculations	14
3.3	Yield Rates versus Percentage Active-Interposer	14
4.1	Comparison of Interposer NoC Topologies	24
5.1	NoC simulation parameters	28
5.2	Full-System Configuration	29
5.3	PARSEC Benchmark Characteristics	33
5.4	Peak Network Operating Frequency	37

List of Figures

2.1	Conventional NoC	5
2.2	NoC Topologies for 64-node Systems	5
2.3	Virtex-7 2000T FGPA	6
2.4	AMD's HBM System	6
2.5	64-core Interposer System with DRAM Stacks	7
3.1	300mm wafers - Chip sizes versus yield	10
3.2	Average number of 64-core SoCs per wafer per 100MHz bin	12
3.3	Comparison of Multi-Socket and MCM RISC Microprocessor Chip Sets	13
3.4	Normalized cost versus Execution Time	15
4.1	Proposed 2.5D multi-chip system	18
4.2	Side-View of Conventional and Proposed Design	19
4.3	Link utilization for a single horizontal row	20
4.4	Baseline Topologies for the interposer	21
4.5	Perspective and side views of concentration and misalignment	21
4.6	Misaligned interposer NoC Topologies	22
4.7	ButterDonut Topologies	23
4.8	Active versus Passive Interposer Implementation	25
5.1	Average packet latency for different topologies	31
5.2	Average packet latency results - SynFull	32
5.3	Distribution of message latencies	32
5.4	Normalized Runtime For Full-System Simulations	33
5.5	Latency and Saturation throughput	34
5.6	Latency & Saturation throughput - Separated memory & coherence traffic	35
5.7	Routing Protocols	36
5.8	Power and area results in 45nm normalized to mesh	36
5.9	Delay-Cost Comparison	38
5.10	Impact of clock crossing latency on average packet latency with 16-core die	39
5.11	Load-latency curves for a monolithic chip	39

Chapter 1

Introduction

As computers require more and more performance, it becomes imperative to design faster and cheaper computing platforms. One of the biggest applications of computers today is their use in data-centers and server farms. Here, they are constantly running near peak performance and improving the speed or energy efficiency, even by a small margin, can have a large impact in the long run. Two important factors that we would like to mention here are that these high-performance systems typically have a large number of computing cores (on the same chip, on the same mother-board, or even on different machines) and large amounts of memory to cater to user demands. One evolving technology that might be able to cater to these growing demands is the use of 3D stacking to fit more computational logic inside a single chip. This has the advantage of occupying less area but also providing faster access between components (the vertical distance to traverse would be much smaller than crossing the length of the chip). However, at the moment, 3D stacking has several obstacles which prevents laying out multiple chips on top of each other in a 3-dimensional stack, the main one being thermal issues¹. A cheaper and more feasible way of approaching multi-chip integration is using a silicon interposer to connect each of the components. This approach is known as 2.5D stacking.

A silicon interposer is essentially a large piece of silicon die with transistors and metal layers that serves as a base for the package. Multiple chips are placed face-down (with the metal layers of the chip facing the metal layers of the interposer) on the interposer. The chip is then connected to the interposer through micro-bumps (μ bumps). The wiring layer on the interposer is used to make chip-to-chip connections. Using an interposer does not prevent the use of 3D stacking. In fact, it is possible to place a 3D stack on top of an interposer. One such use case is to place multiple 3D-stacked DRAM on top of an interposer and interface them with a computing core. In this work, we consider such a baseline system: a 64-core chip with four 3D-DRAM stacks integrated using a silicon interposer.

¹ When multiple chips are stacked, the effective surface area does not grow at the same rate as the number of logical components. Since the rate of heat dissipation is proportional to the surface area and the heat generated is proportional to the number of working transistors, 3D stacking does not scale well.

1.1 Research Highlights

In this work:

- We make some key observations regarding silicon-interposer-based stacking. In particular, we note that it leaves a large amount of area on the interposer under-utilized.
- We propose systems which take advantage of the unused interposer fabric to try to improve performance over existing 2.5D systems.
- We consider disintegrating large monolithic chips into smaller chips and consider the interposer as a potential candidate for integrating them. We then do a cost versus performance evaluation of disintegrated chips on an interposer when compared against a monolithic core.
- We propose several new network topologies which can take advantage of the silicon interposer to provide increased bandwidth and performance over more conventional network topologies. One of the key contributions is our proposal of “misaligned” topologies which better cater to multi-chip designs.

This work has lead to a publication in the International Symposium on Microarchitecture (MICRO-48) [35].

1.2 Organization

This thesis is divided into seven chapters including this one. In Chapter 2, we look at the background of networks on chip, die stacking, and silicon interposers. We then motivate the work in Chapter 3. In Chapter 4, we show the baseline design we assumed as a starting point. We then look at other possible topologies which could have a large positive effect on the system. In Chapter 5, we first describe the methodology we used to evaluate our designs. We then evaluate our designs using this methodology and provide the results and an analysis. In Chapter 6, we look at other works that related to the ideas that we present in this thesis. Finally, Chapter 7 finally provides the closing arguments to this work as well as future directions that this work might take us.

Chapter 2

Background

As computer architects, we are continuously seeking to create the next generation of computing platforms with one or two aims: improving performance and/or power efficiency. One key trend that has pushed computing devices forward is the scaling down of transistor sizes, allowing us to pack more transistors and hence more logic within the same area. Industry has pushed to keep the scaling of transistors to be in line with Moore's law, i.e. *every 18 months, chip performance would double*, basically implying that transistor scaling would allow us to pack twice as many transistors within the same chip area.

Reducing transistor size has the added benefit of reducing the gate delay. This can lead to an increased clocking frequency. According to Borkar [14], scaling a design can reduce the gate delay and the lateral dimensions by 30% which can lead to a frequency improvement of $1.43\times$, with no increase in the power dissipation. However, it does show an increase in power if we wish to take advantage of the additional area. To scale this increase in power dissipation back down, typically the supply voltage of the new process node is reduced by 30%. For the original design, this can show a decrease in power by 50%. Thus, this allows us to use the extra transistor logic at no energy cost. This is essentially summed up by Dennard's law [23] which, in essence, states that the transistor power density remains constant with different process nodes.

However, around the years of 2005-2007, Dennard's law started breaking down [12]. This was due to certain assumptions that Robert Dennard made regarding MOSFET scaling which did not hold any longer. This meant that we could no longer scale transistors down and utilize all of the additional transistors without sacrificing an increase in power consumption. Another key result of Dennard's paper [23] was that scaled interconnect does not speed up, i.e. it provides roughly constant RC delays. Initially, the wire delays were a fraction of the critical path. However, they have become a key component, affecting the peak operating frequency [12, 13]. For uni-processors, frequency scaling was one of the key factors leading to performance improvement in each generation. This, in addition to thermal considerations, led to the inability to scale uni-processor core frequency past a point. To address these issues, multi-core processor designs were proposed and they soon became the norm. Being able to utilize the additional transistors, multi-core designs could improve upon earlier designs and show increased net throughput, without having to increase the clocking frequency.

2.1 Network on Chip

Modern Chip Multi-Processors (CMP) use buses to allow communication between the different cores on chip. However, buses do not scale well to larger number of cores, due to the contention that the bus faces when many or all cores request for it. This led the way to several new architectures to scale CMPs to a large number of cores. One such method is to route core-to-core and core-to-memory traffic using an interconnection network on chip [18]. A conventional Network-on-Chip is shown in Figure 2.1.

A network-on-chip (NoC) replaces dedicated point-to-point links as well as the global bus with a single network. Network clients could be general purpose processors, GPUs, DSPs, memory controllers, or any custom logic device. Each client has a Network Interface (NI) which is connected to a network router (indicated as *R* in Figure 2.1). If a client wants to communicate with another, it sends a packet into the network which subsequently gets routed to the appropriate destination router. The router finally sends it to the destination client through the Network Interface.

NoCs offer many advantages over conventional methods of on-chip communication [18]:

- On-chip wiring resources are shared between all cores. This improves the efficiency of the area used for wiring.
- NoCs enable better scaling of multi-core processor designs.
- The wiring has a more regular structure. This allows for better optimization of the electrical properties which can result in less cross-talk.
- NoCs also promote modularity; the interfaces can be standardized. For example, we can have a standard router design and network interface, allowing easier integration of various IP cores (Intellectual Property Cores).

2.1.1 Network Parameters and Metrics

Before getting into the details about on-chip interconnection networks, it is imperative to define the parameters that are used to define networks as well as a few metrics that can be used to measure the network performance.

Flits: One technique used to improve network performance is known as wormhole flow control [2]. In this method, packets are divided into smaller segments known as flits. Flits use the same path and move sequentially through the network. The flit width is the number of bits per each flit and is usually equal to the width of the physical link.

Hop count: A single hop is defined as when a flit moves from one router to another. Hop count is similarly defined on a flit-level and is the total number of hops a flit makes from the starting node to its destination. We can define the average hop count for the entire network by considering all pairs of sources and destinations. A lower average hop count can imply (provided other parameters are similar) that the network is better connected and flits can reach their destinations faster.

Network Diameter: The network diameter is defined as the longest minimal-hop between any source-destination pair in the network. For example, the network diameter of the mesh network shown in Figure 2.1 is six: the path from the north-west corner to the south-east corner.

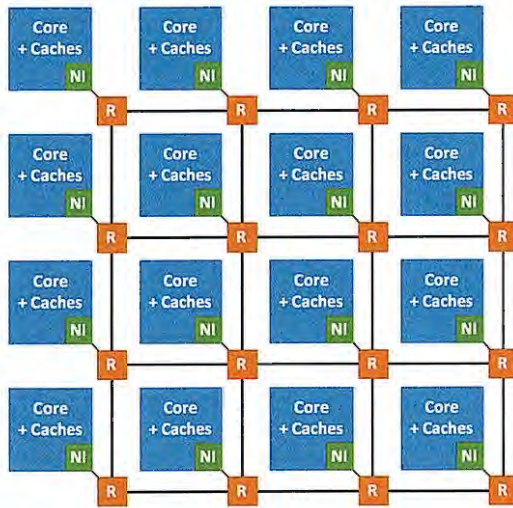


Figure 2.1: Conventional Network on Chip

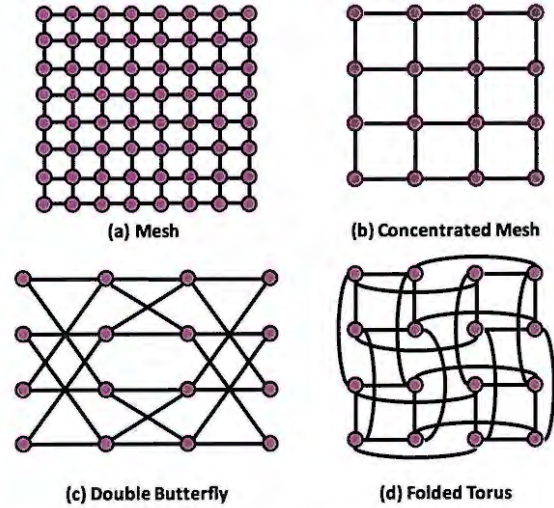


Figure 2.2: NoC Topologies for 64-node Systems

Bisection Bandwidth: The bisection bandwidth is the total bandwidth across a central cut (such that the two sides are equally divided) in the network. A higher bisection bandwidth is directly correlated to improved network performance.

Network Latency: Network latency is the average time a packet spends in the network considering all source-destination pairs. Similarly, average packet latency is the average time a packet takes to reach the destination from the source. The subtle difference between the two is that average packet latency includes any stalls that a packet may face in the injection queue at the node. The packet latency is higher or equal to the network latency.

2.1.2 Network Topology

The arrangement of network clients within the NoC is defined as the network topology. The topology determines many characteristics of the network including the number of ports per router (known as the radix), the bisection bandwidth, channel load, and the path delay [2]. Some examples of different network topologies are shown in Figure 2.2. Each of these topologies contain 64 nodes. Figure 2.2(a) shows a mesh in which there is a one router dedicated to each node.

Concentration: Concentration is a useful technique that improves the utilization of the physical links by connection multiple nodes to a single router on the system. The number of nodes connected to a single router is known as the concentration factor. Figure 2.2(b, c, d) are concentrated networks with a concentration factor of 4. The original 64 router network is reduced to just 16 routers due to the concentration. The bandwidth and the physical link width for each router remains the same and is shared between the four nodes attached to it. Additionally, their longer links result in lower average hop count.

2.2 Die-Stacking

Moore's Law has conventionally been used to increase integration. In recent years, fundamental physical limitations have slowed down the rate of transition from one technology node to the next, and the costs of new fabs are sky-rocketing. However going forward, almost everything that can be easily integrated has already been integrated! What remains is largely implemented in disparate process technologies (e.g., memory, analog) [10]. This is where the maturation of die-stacking technologies comes into play. Die stacking enables the continued integration of system components in traditionally incompatible processes. Vertical or 3D stacking [60] takes multiple silicon die and places them on top of each other. Inter-die connectivity is provided by through-silicon vias (TSVs).

2.5D stacking [22] or *horizontal stacking* is another approach to die stacking as an alternative to vertical stacking. In this approach, multiple chips are combined by using stacking them all on top of a single base silicon interposer. The base interposer is a regular but larger silicon stop with the conventional metal layers facing upwards. Current interposer implementations are passive, i.e. they do not provide transistors on the interposer silicon layer. Only metal routing between chips and TSVs for signals entering/leaving the chip [50] are provided. This technology is already supported by design tools [31], is already in some commercially-available products [50, 53], and is planned for future GPU designs [26]. One recent application is AMD's High-Bandwidth Memory [1] which combines 3D-stacked memory with a CPU/GPU SoC Die with an interposer. This is shown in Figure 2.4. Another example is shown in Figure 2.3. Future generations could support *active interposers* (perhaps in an older technology) where devices could be incorporated on the interposer.

With 2.5D stacking, chips are typically mounted face down (in a flip-chip design) on the interposer with an array of micro-bumps (μ bumps). Current micro-bump pitches are 40-50 μ m, and 20 μ m-pitch technology is under development [27]. The μ bumps provide electrical connectivity from the stacked chips to the metal routing layers of the interposer. Die-thinning¹ is used on the interposer for TSVs to route I/O, power, and ground to the C4 bumps (which connect the interposer to the substrate). The interposer's metal layers are manufactured with the same back-end-of-line process used for metal interconnects on regular "2D" stand-alone chips. As such, the intrinsic metal density and physical characteristics (resistance, capacitance) are the same as other on-chip wires. Chips stacked horizontally on an interposer can communicate with each other with point-to-point electrical connections from a source

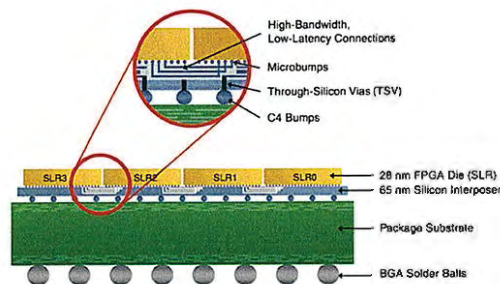


Figure 2.3: Virtex-7 2000T FPGA Enabled by SSI Technology [50]

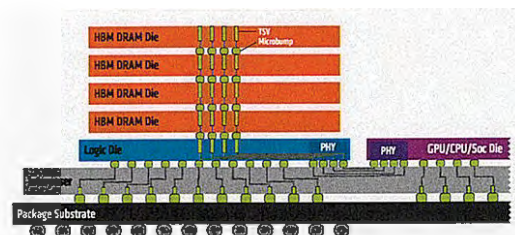


Figure 2.4: AMD's High Bandwidth Memory System [1]

¹Most wafers start off with a thickness of around 1200 μ m. This provides mechanical stability during the fabrication process. Die-thinning is done post-fabrication in some cases where slim packages (with a smaller height profile) are required.

chip's top-level metal, through a micro-bump, across a metal layer on the interposer, back through another micro-bump, and finally to the destination chip's top-level metal. Apart from the extra impedance of the two micro-bumps, the path from one chip to the other looks largely like a conventional on-chip route of similar length. As such, unlike conventional off-chip I/O, chip-to-chip communication across an interposer does not require large I/O pads, self-training clocks, advanced signalling schemes, etc.

2.2.1 2.5D vs. 3D Stacking

The two stacking styles have their own set of advantages and disadvantages. 3D stacking potentially provides more bandwidth between chips. The bandwidth between two 3D-stacked chips is a function of the chips' common surface area. 3D stacking also incurs additional area for their TSVs due to increased tensile stress around them. This causes variation in the carrier mobility in the neighborhood of the TSVs, often requiring large "keep-out" regions to prevent nearby cells from being affected [3, 46]. On the other hand, the bandwidth between 2.5D-stacked chips is bound by their perimeters. Additionally, the 2.5D-stacked chips are flipped face down on the interposer so that the top-layer metal directly interfaces with the interposer micro-bumps and therefore, do not require TSVs on the individual chips themselves.

Another limitation of vertical (3D) stacking is that the size of the processor chip limits how much DRAM can be integrated into the package. Each subsequent chip is typically of the same or smaller size. With 2.5D stacking, the capacity of the integrated DRAM is limited by the size of the interposer rather than the processor. The chips can also have large variability in dimensions.

For multi-core SoC designs, 2.5D stacking is compelling because it does not preclude 3D stacking. In particular, 3D-stacked DRAMs may be used, but instead of placing a single DRAM stack directly on top of a processor, stacks may be placed next to the processor die on the interposer. For example, Figure 2.5 shows a 2.5D-integrated system with four DRAM stacks on the interposer. Using the chip dimensions assumed in this work (Section 4.1), the same processor chip with 3D stacking could only support two DRAM stacks (i.e., half of the integrated DRAM capacity). Furthermore, directly stacking DRAM on the CPU chip could increase the engineering costs of in-package thermal management [17, 28, 49].

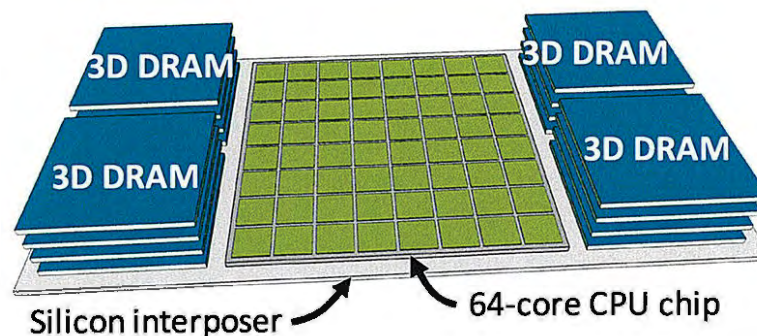


Figure 2.5: An example interposer-based system integrating a 64-core processor chip with four 3D stacks of DRAM.

2.3 Silicon Interposers and Their Networks

Using 2.5D stacking to design a multi-core SoC introduces several interesting opportunities. One of the most important opportunities (with regards to this thesis) is how to interconnect different chips. In a monolithic chip, a network on chip could be used to interface different components on the chip such as the different CPUs, caches and memory controllers. However, it is possible that these components are distributed across different chips. With the wiring resources on the interposers, we now have the chance to design a new set of networks catering specifically to multi-chip designs. We deal with this further in Chapter 3.

Chapter 3

Motivation

The increasing core counts of multi-core (and many-core) processors demand more memory bandwidth to keep all of the cores fed with operable data. Die stacking can address the bandwidth problem while reducing the energy-per-bit cost of accessing memory. A key initial application of die-stacking is silicon interposer-based integration of multiple 3D stacks of DRAM, shown in Figure 2.5 [10, 45, 24], potentially providing several gigabytes of in-package memory¹ with bandwidths already starting at 128GB/s (per stack) [32, 41].

The performance of a multi-core processor is not only limited by the memory bandwidth, but also by the bandwidth and latency of its NoC. The inclusion of in-package DRAM must be accompanied by a corresponding increase in the processor’s NoC capabilities. However, increasing the network size, link widths, and clock speed all come with significant power, area, and/or cost overheads for additional metal layers. The presence of an interposer to interact with other chips (which provides “free” additional area in terms of logic and wiring) presents several opportunities which can help us achieve increased performance on the NoC and reduce overall costs. The interposer also allows us to integrate more resources into one package than is possible with one chip.

In this chapter, we first look at the costs and potential benefits of breaking a large chip into smaller chips in Section 3.1. In Section 3.2, we consider how these smaller chips can be combined to replicate the functionality of the monolithic chip (e.g. Four 16-core multi-chip system versus a 64-core chip). Finally, in Section 3.3, we do a preliminary analysis of the costing of doing this, as well as present the research problem that this thesis deals with.

3.1 Chip Disintegration

Manufacturing costs of integrated circuits are increasing at a dramatic rate. The cost of a chip scales with its size. A larger chip’s high cost comes from two sources:

Geometry: The *geometry* of a larger chip lets fewer chips fit on a wafer. Figure 3.1 shows two 300mm wafers. Figure 3.1(a) is filled with $297mm^2$ chips whereas Figure 3.1(b) is filled with $148.5mm^2$ chips. 192 larger chips can fit on a single wafer for a total area utilization of $5.70 \times 10^4 mm^2$. The smaller chips

¹Several gigabytes of memory is unlikely to be sufficient for high-performance systems and will likely still require tens or hundreds of gigabytes of conventional (e.g., DDR) memory outside of the processor package. Management of a multi-level memory hierarchy is not the focus of this work.

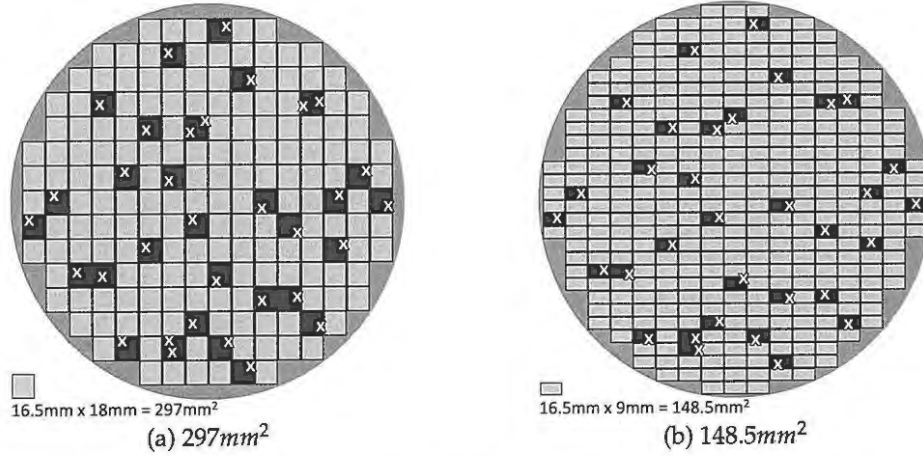


Figure 3.1: Example 300mm wafers with two different chip sizes showing the overall number of chips and the impact on yield of an example defect distribution.

can be packed more tightly (using the area around the periphery of the wafer). This results in 395 chips per chip ($5.87 \times 10^4 \text{mm}^2$) which is a 3% increase in the total computational area.

Manufacturing Defects: Larger chips are more prone to manufacturing defects. Defects can appear on the wafer during the manufacturing process. They are not dependent on the size of the die. If a defect is present within the boundaries of the die, it renders the die inoperable. For a large die, a single defect wastes more silicon than when it kills a smaller die. Figure 3.1 shows an example distribution of defects on two wafers that renders some fraction of the chips inoperable. We used Monte Carlo simulations (using defect rates from manufacturing datasheets) to simulate defects on both the chip sizes. For the average case, this reduces the 192 original large die to 162 good die per wafer (GDPW), resulting in a ~16% yield loss. For the half-sized die, we go from 395 die to 362 GDPW for a ~8% yield loss. In general, *a smaller chip gets you more chips, and more of them work.*

3.1.1 Chip Cost Analysis

Smaller chips may be cheaper but they also provide less functionality. For example, a dual-core chip (ignoring caches for now) may take half as much area as a quad-core chip. The natural line of thought might lead one to the question “can you just replace larger chips with combinations of smaller chips?” Assuming that is possible to do so, we could have the functionality of a larger chip while maintaining the economic advantages of smaller chips. We make use of analytical yield models with a fixed cost-per-wafer assumption and automated tools for computing die-per-wafer [25] to consider a range of defect densities. We can assume a 300mm wafer and a baseline monolithic 64-core die of size 16.5mm $\times$ 18mm (the same assumption is used in the recent interposer-NoC paper [24]). Smaller-sized chips can be derived by halving the longer of the two dimensions (e.g., 32-core chip is 16.5mm $\times$ 9mm). The yield rate for individual chips is estimated using a simple classic model [54]:

$$\text{Yield} = \left(1 + \frac{D_0 A_{\text{crit}}}{\alpha} \right)^{-\alpha}$$

where D_0 is the defect density (defects per m^2), n is equal to the number of vulnerable layers (13 in our analyses, corresponding to one layer of devices and 12 metal layers), A_{crit} is the total vulnerable area (i.e., a defect that occurs where there are no devices does not cause yield loss), and α is a clustering factor to model the fact that defects are typically not perfectly uniformly distributed. We ran our experiments for several other values of α , but the overall results were not qualitatively different. For A_{crit} , we assume different fractions of the total chip area are critical depending on whether it is a device or metal layer. Table 3.1 summarizes the impact of implementing a 64-core system ranging from a conventional 64-core monolithic chip all the way down to building it using 16 quad-core chips. The last column shows the final impact on the number of good SoCs we can obtain per wafer. It should be noted that the exact parameters here are not crucial: the main result (which is not new) is that smaller chips are cheaper.

Cores Per Chip	Chips Per Wafer	Chips per Package	Area per Chip (mm^2)	Chip Yield	Good Die Per Wafer	Good SoCs per Wafer
64	192	1	297.0	84.5%	162	162
32	395	2	148.5	91.7%	362	181
16	818	4	74.3	95.7%	782	195
8	1,664	8	37.1	97.8%	1,627	203
4	3,391	16	18.6	98.9%	3,353	209

Table 3.1: Example yield analysis for different-sized multi-core chips. A “SoC” here is a 64-core system, which may require combining multiple chips for the rows where a chip has less than 64 cores each.

The results in Table 3.1 assume the usage of known-good-die (KGD) testing techniques so that individual chips can be tested before being assembled together to build a larger system. If die testing is used, then the chips can also be speed-binned prior to assembly. We used Monte Carlo simulations to consider three scenarios:

1. A 300mm wafer is used to implement 162 monolithic good die per wafer (as per Table 3.1).
2. The wafer is used to implement 3,353 quad-core chips, which are then assembled without speed binning into 209 64-core systems.
3. The individual die from the same wafer are sorted so that the fastest sixteen chips are assembled together, the next fastest sixteen are combined, and so on.

We can simulate the yield of a wafer by starting with the good-die-per-wafer based on the geometry of the desired chip (Table 3.1). For each quad-core chip, we randomly select its speed using a normal distribution (mean 2400MHz, standard deviation of 250MHz). Our simplified model treats a 64-core chip as the composition of sixteen adjacent (4×4) quad-core clusters, with the speed of each cluster chosen from the same distribution as the individual quad-core chips. Therefore the clock speed of the 64-core chip is the minimum from among its constituent sixteen clusters. At this point, we do not model the slow-down due to the interposer-integration but we merely look at fast we can run the multiple chips. For each configuration, we simulate 100 different wafers worth of parts, and take the average over the 100 wafers. Similar to the yield results, the exact distribution of per-chip clock speeds is not so critical: so long as there exists a spread in chip speeds, binning and reintegration via an interposer can potentially be beneficial for the final product speed distribution².

Figure 3.2 shows the number of 64-core systems per wafer in 100MHz speed bins, averaged across

² Expenses associated with the binning process are not included our cost metric, as such numbers are not readily available, but it should be noted that the performance benefits of binning could incur some overheads. Similarly, disintegration into a larger number of smaller chips requires a corresponding increase in assembly steps, for which we also do not have relevant cost information available.

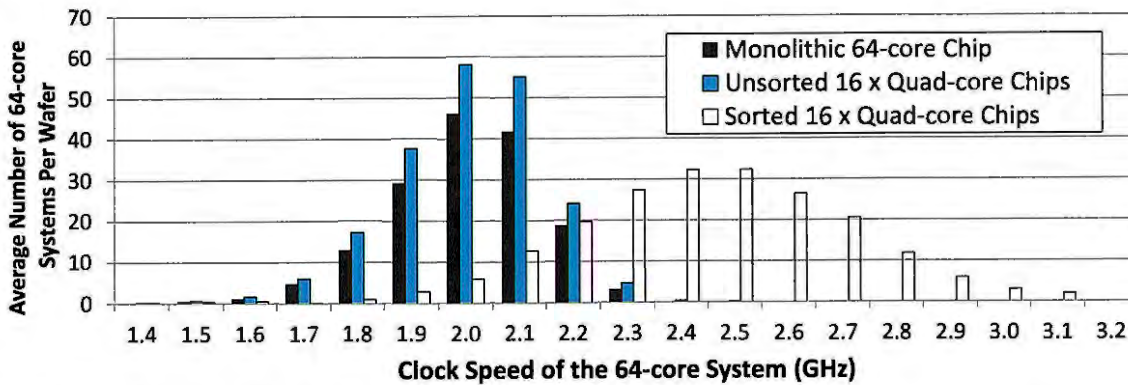


Figure 3.2: Average number of 64-core SoCs per wafer per 100MHz bin from Monte Carlo simulations of 100 wafers.

one hundred wafer samples per scenario. The Monolithic 64-core chip and 16 quad-core approaches have similar speed distributions. However with speed binning, we avoid the situation where overall system speed is slowed down by the presence of a single slow chip, resulting in significantly faster average system speeds (the mean shifts by ~ 400 MHz) and more systems in the highest speed bins (which usually carry the highest profit margins).

3.2 Integration of smaller chips

Now that we see that there is potential for multi-chip systems to be competitive with monolithic cores, let's consider how multiple chips can be combined to form a single system. There are several methods that we consider:

- **Multi-socket:** Symmetric multi-processing (SMP) systems on multiple sockets have existed for many decades now. Chips are placed on different sockets on the same motherboard and connected via high-speed interconnects (e.g. Intel QuickPath interconnect [30]). SMPs using this design methodology share memory (Uniform/non-uniform memory access for the different chips) and can have coherence protocols. The memory modules are provided as DIMMs (Dual In-Line Memory Modules). DIMMs typically have a peak bandwidth rate which is limited by the number of I/O ports on the modules as well as the interconnect. Each chip has a similar limitation due to the number of pins per package. This is the primary disadvantage of this approach; the bandwidth and latency between two chips are limited by number of pins per package.
- **Multi-chip Modules (MCM):** In an MCM, chips are placed horizontally on the package substrate (generally a ceramic flatpack) and connected to the substrate using C4 bumps. Wires between chips are routed using metal layers on the substrate. This allows the designer to pack the chips together much closer than with a multi-socket system, thereby reducing the interconnect delay significantly. Figure 3.3 shows a multi-socket system in comparison to an MCM. While this alleviates the limitations of pin connections, the bandwidth and latency are restricted by the density of C4 bumps and the substrate routing that connects the silicon die.
- **Silicon Interposers:** The silicon interposer, as discussed earlier, is essentially a large chip upon which smaller dies can be stacked. The μ bumps with which chips are connected to the interposer

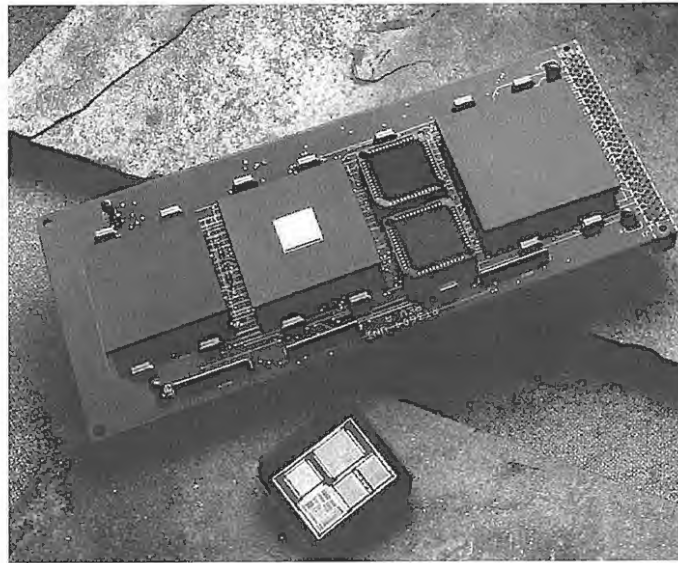


Figure 3.3: Comparison of Multi-Socket and MCM RISC Microprocessor Chip Sets [11]

are denser than C4 bumps ($\sim 9\times$ better). The main disadvantage is having to traverse through the interposer when communicating off-chip.

- **3D stacking:** The different chips could be vertically stacked above one another. Each chip is thinned and implanted with TSVs for vertical interconnects. 3D stacking has the highest potential bandwidth among these four options but also the highest complexity.

The SMP and MCM approaches are less desirable as they do not provide adequate bandwidth for arbitrary core-to-core cache coherence without exposing significant NUMA effects. As such, we do not consider them further. 3D stacking by itself is not (at least at this time) as an attractive of a solution because it is more expensive and complicated, and introduces potentially severe thermal issues. This leaves us with silicon interposers.

3.2.1 Silicon Interposer-Based Chip Integration

Silicon interposers offer an effective mechanical and electrical substrate for the integration of multiple disparate chips. Current 2.5D stacking primarily uses the interposer for connections between adjacent chips (e.g., processor to stacked DRAM) only at their edges. An example of this is shown in Figure 4.2a. Apart from this limited routing, the vast majority of the interposer's area and routing resources are not utilized. In particular, if one has already paid for the interposer for the purposes of memory integration, any additional benefits from exploiting the interposer are practically "free". This area can effectively be used to improve the NoC capabilities to enable better use of the increased memory bandwidth.

There are two design approaches when looking at interposer-based designs. The first method is use the interposer purely for the purpose of wiring (by only using the metal layers on the interposer). Any extra routers which are required by any new network topologies will have to reside on the chip. This is known as a passive interposer. Current designs [50, 53] use this approach. Passive interposers contain no devices, only routing. The primary disadvantage to this approach is that for a series of hops through the interposer, a packet needs to pass through a pair of μ bumps for each hop. However, due to the interposer having a low critical area (A_{crit}), the resulting yield (of the interposer) is very high.

Parameter	Value
n	13
$Frac_{crit}$ (wire)	0.2625
$Frac_{crit}$ (logic)	0.7500
α	1.5

Table 3.2: Parameters for the chip yield calculations

D_0	500	1000	1500	2000	2500
Passive	98.5%	97.0%	95.5%	94.1%	92.7%
Active 1%	98.4%	96.9%	95.4%	93.9%	92.5%
Active 10%	98.0%	96.1%	94.2%	92.4%	90.7%
Fully-active	87.2%	76.9%	68.5%	61.5%	55.6%

Table 3.3: Yield rates for $24\text{mm} \times 36\text{mm}$ interposers varying the active devices/transistors from none (passive) to 100% filled (fully-active) across different defect rates (D_0 in defects per m^2)

The alternative to this approach is to use an active interposer, i.e. place both the wires and the router logic on the interposer. This design makes use of both the metal layers and the transistors on the interposer. This approach enables much more interesting NoC organizations. An active interposer provides a lot more logic that the designer can use. For regular chips, a good design would typically attempt to maximize the functionality by cramming in as many transistors as available on the chip's budget. However, making more complete use of the interposer would lead to a high A_{crit} multiplied over a very large area. This would lead to low yields and high cost, resulting in the same problem we are trying to solve. However, in the design of an NoC spanning the silicon interposer, there is no need to use the entire interposer. The geometry of the design on the interposer is dependent on the layout of the chips and memory stacked upon it. As such, we advocate for a *Minimally-active Interposer* - implement the devices required for the functionality of the system (i.e. the routers and repeaters) on the interposer but nothing more. This will result in a sparsely-populated interposer with a lower A_{crit} and thus a lower cost. Another factor that affects the yield of the interposer is the fact that we propose to use an older process node to manufacture the interposer. For example, for a 14nm or 22nm process, we could use 32nm or 45nm for the interposer. This has the advantage of having a higher yield and lower manufacturing costs. Additionally, since the process generations are close, the capacitance per unit length will not differ much between the two technology nodes. The capacitance per unit length for the 65nm node for the M1 layer is 0.168 pF/mm whereas it is only 0.157 pf/mm for the 45nm node [57, 16, 51]. With respect to a 22nm process where the capacitance per unit length is 0.129 pF/mm, the difference is 30% for the 65nm case as compared to just 20% for the 45nm case and even smaller if we implement the interposer in a 32nm process. We are also not targeting an aggressive clock frequency for the network- this is common with NoCs where the network will often run slower than the processors. Thus, meeting timing with an older process generation will not be an issue.

We perform yield modelling to estimate the yields of the different interposer options: a passive interposer, a minimally-active interposer, and a fully-active interposer. We assume the size of the interposer to be $24\text{mm} \times 36\text{mm}$ (864mm^2) with six metal layers on the interposer. For the passive interposer, the A_{crit} for the logic is zero and non-zero for the metal layers. For a fully-active interposer, we use the same A_{crit} as shown in Table 3.2. For a minimally-active interposer, we estimate that the total interposer area needed to implement our routers (logic) and links (metal) to be only 1% of the total interposer area. To have a conservative estimate, we also consider a minimally-active interposer where we pessimistically assume the router logic consumes $10\times$ more area. Minimizing utilization of the interposer for active devices also minimizes the potential for undesirable thermal interactions resulting from stacking highly active CPU chips on top of the interposer.

Table 3.3 shows the estimated yield rates for the different interposer options. This uses the same defect rate (2000 defects/ m^2) from Table 3.1 as well as four other rates. The two lowest rates reflect that the interposer is likely to be manufactured in an older, more mature process node with lower defect

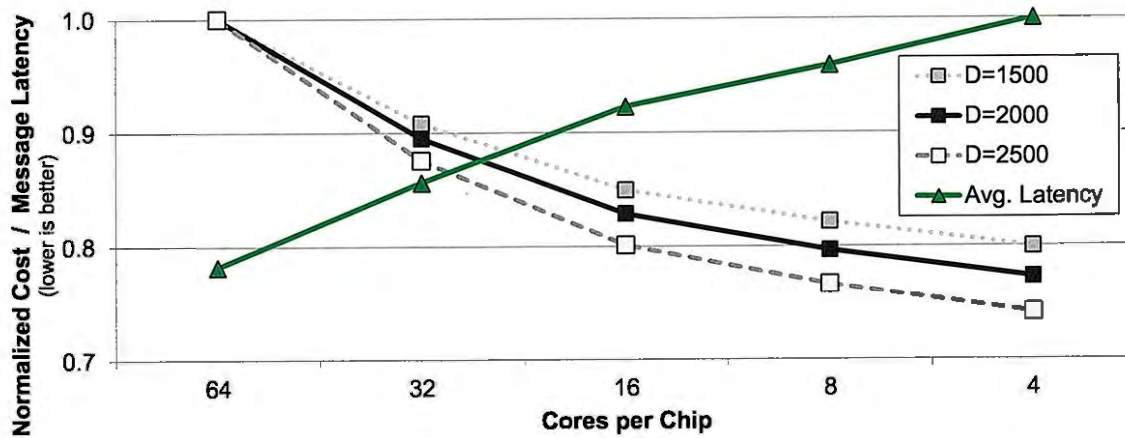


Figure 3.4: Normalized cost and execution time (lower is better for both) for different multi-chip configurations. 64 cores per chip corresponds to a single monolithic 64-core die, and 4 cores per chip corresponds to 16 chips, each with four cores. Cost is shown for different defect densities (in defect/m²), and the average message latency is normalized to the 16 quad-core configuration.

rates. The passive interposer has a non-perfect yield rate ($<100\%$) as it still uses a metal layers that can be rendered faulty by manufacturing defects. For a fully-active interposer, higher defect rates (1,500-2500 defects/m²) result in very low yields. This is not surprising given that a defect almost anywhere on the interposer could render it a loss. This is the primary reason why one would likely be skeptical of active interposers. However, Table 3.3 shows that when using only the minimum amount of active area necessary on the interposer, the yield rates are not very different from the passive interposer. The vast majority of the interposer is *not* being used for devices; defects that occur in these “white space” regions do not impact the interposer’s functionality. So even with the conservative assumption that the NoC routers consume 10% of the interposer area at the highest defect rates considered, our model predicts yields of over 90%. As a result, we believe that augmenting an otherwise passive interposer with just enough logic to do what you need has the potential to be economically viable, and it should be sufficient for NoC-on-interposer applications.

3.2.2 Limitations of Cost Analysis

It is important to note that the above yield models cannot replace a complete cost analysis. However, the lack of publicly available data makes it incredibly difficult to provide meaningful dollar-for-dollar cost comparisons. Factors such as additional costs for the extra masks (mask set costs are effectively amortized over the all units shipped) for an active interposer and additional processing steps (incurred per unit) must be combined with the yield analysis to arrive at a final decision as to whether a given SoC should use an active interposer.

3.3 The Research Problem

Taking the cost argument *alone* to its logical limit would lead one to falsely conclude that a large chip should be disintegrated into an infinite number of infinitesimally small die. The countervailing force

is performance. While breaking a large system into smaller pieces may improve overall yield, going to a larger number of smaller chips increases the amount of chip-to-chip communication that must be routed through the interposer. In an interposer-based multi-core system with a NoC distributed across chips and the interposer, smaller chips create a more fragmented NoC resulting in more core-to-core traffic routing across the interposer, which eventually becomes a performance bottleneck. Figure 3.4 shows the cost reduction for three different defect rates, all showing the relative cost benefit of disintegration. The figure also shows the relative impact on performance³. So while more aggressive levels of disintegration provide better cost savings, it is directly offset by a reduction in performance.

The problem we explore is how we can get the cost benefits of a disintegrated chip organization while providing an NoC architecture that still behaves similarly in terms of performance to one implemented on a single monolithic chip. We aim to do this by utilizing the additional resources available on the interposer to design networks specifically for 2.5D systems instead of just using edge-to-edge connections. These new networks will utilize resources on both the interposer and on each of the chips. This allows for a variety of optimizations which can help reduce the average number of hops between different cores as well as to memory. We can also route packets in such a way so as to distribute the network load across the resources on and off-chip. We discuss the challenges and our proposal to address them in Chapter 4.

³ We show the average message latency for all traffic (coherence and main memory) in a synthetic uniform-distribution workload, where CPU chips and the interposer respectively use 2D meshes vertically connected through μ bumps. See Chapter 5 for full details.

Chapter 4

NoC Architecture

4.1 Baseline Architecture

The baseline design that we serve as the starting point of our design is shown in Figure 4.1. This architecture is a 2.5D system with four 16-core CPU chips. The interposer in this design is relatively large but fits within an assumed reticle limit of $24\text{mm} \times 36\text{mm}$. On the interposer are four 16-core chips as well as four 3D DRAM stacks. Each chip is of the size $7.75\text{mm} \times 7.75\text{mm}$. The DRAM stacks (each of size $8.75\text{mm} \times 8.75\text{mm}$) are placed on either side of the multi-core die. Each of these four stacks are assumed to have a size similar to a JEDEC Wide-IO DRAM [33, 37]. Each stack has four channels¹, with 16 channels in total. The chip-to-chip spacing is assumed to be 0.5mm for all pairs shown in the figure.

Current 2.5D designs utilize the interposer for chip-to-chip routing and for vertical connections to the package substrate for power, ground, and I/O [53]. In current industry designs, the interposer is used minimally. Therefore, we decided to use a passive interposer for the baseline design. In this design, the DRAM stacks are integrated with four 16-core multi-core chips on a passive interposer. There are only edge-to-edge connections between the multiple chips as shown in Figure 4.2a.

Our proposal seeks to make use of the unused routing resources available on the interposer layer to implement a system-level NoC. This concept is illustrated in Figure 4.2b. Thus, in addition to the on-chip network, there is a secondary (logical) 10×8 mesh network on the interposer which connects the various chips as well as the four DRAM stacks where each core has its own link into the interposer. Each core in each of the four chips has a connection (through μbumps) to the interposer layer, totalling 16 connections per each multi-core chip and four connections for each DRAM stack (one for each memory channel).

4.2 Routing Protocol

When a core wants to communicate off-chip (either with a core on another chip or with memory), it has to use the interposer network or a combination of the on-chip and the interposer network to reach its destination. There are many cases where there are multiple possible paths that a packet could take from source to destination. The quality of a path can be measured by the number of hops required for a packet to be routed from the source to the destination. A lower hop count implies that a packet

¹Having multiple channels for each DRAM memory stack increases the maximum bandwidth of the memory module.

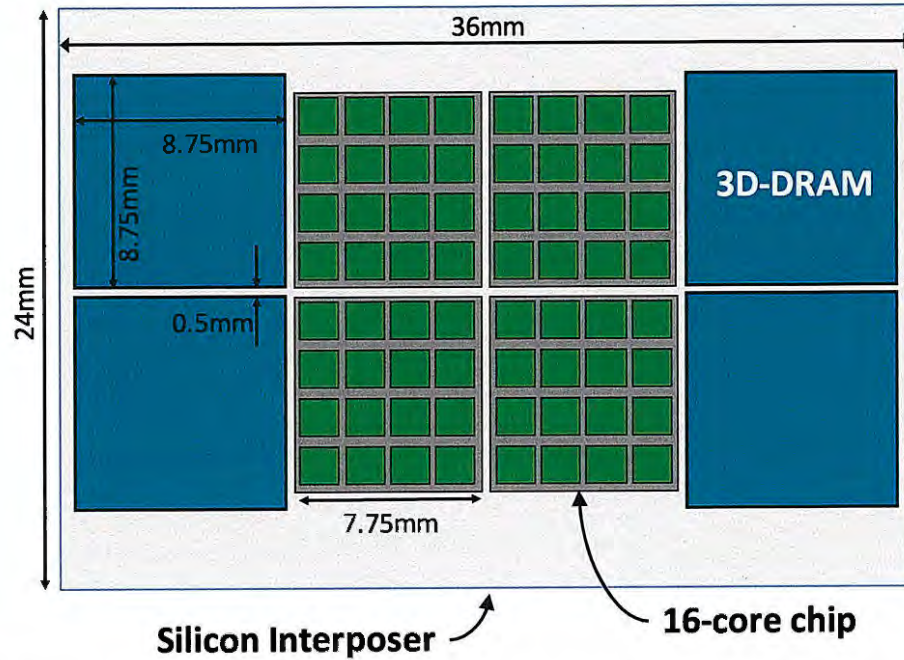


Figure 4.1: Top View of evaluated 2.5D multi-core system with four DRAM stacks placed on either side of the processor dies

can reach the destination in fewer steps. The hop count directly influences the average packet latency which is a good indicator of the network performance. For a given network, varying the routing protocol can have an effect on the system performance. In most cases, we can statically determine the minimal paths (in terms of hop count and latency). However, in certain cases, there are multiple minimal paths. Therefore, it becomes imperative to specify a routing protocol which can be used to resolve such ambiguities.

For standard 2D mesh-based NoCs, the most common routing protocol is Dimension Order Routing (DOR). DOR can either be XY or YX for 2D networks. DOR-XY first routes a packet along the horizontal links (through the shortest path) to the appropriate column. It then routes the packet vertically (again, along the shortest path) until it reaches its destination. DOR-YX routes vertically first and then horizontally. When we look at 2.5D networks, we are adding a third dimension. There are two ways we can look at these networks. The first is to look at the system as two separate NoCs and provide a routing protocol for each independently and have an overseeing protocol which controls when a packet switches between the two networks. The second is to consider the network as a 3-dimensional network with the vertical links constituting the Z-axis. For simpler network topologies, the second approach is simpler since it does not require any modification of earlier protocols. However, we will use the first method since it allows finer control of resource utilization on the interposer and on chip. This allows us to specify the routing protocol for the on chip network, the interposer network, and when (and where) packets switch between the two sub-networks.

For the on-chip component, all architectures that we use in thesis are a standard mesh and thus we use simple DOR-XY routing. For the interposer component, mesh-based topologies use DOR-XY. The double butterfly uses extended destination tag routing [2]. The ButterDonut topology (Subsection 4.3.2) uses table-based routing.

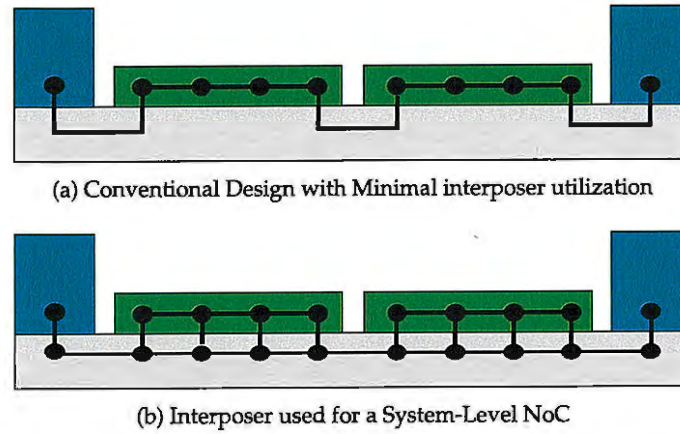


Figure 4.2: Side-View of Conventional and Proposed Design

In all of the topologies we use, we attempt to minimize the number of hops for every pair of nodes in the system. The last thing that we have to address is when to use the on-chip network and when to inject a packet into the interposer network. When a core wants to communicate with another core on the same chip, we restrict it by forcing it to stay on chip. When a core communicates with a memory controller, we always go to the interposer network- the packet gets injected into the router attached to the core which then uses the μ bump to move it into the interposer network. It subsequently gets routed on the interposer network to the destination. The reverse occurs when the memory controller responds to this request: the packet is injected into the interposer network by the memory controller, gets routed on the interposer network, traverses up the μ bump to the router connected to the core and finally to the core. When routing between two cores in different chips, there are two approaches that we consider:

- **Interposer-First Routing:** With this approach, if a packet is to go off-chip (to another core), after being injected into the router connected to the node, it traverses through the μ bump to the interposer layer. It takes the most optimal route to the destination on the interposer layer itself, finally going up the μ bump to the destination router.
- **Minimal Interposer Routing:** In this approach, for core-to-core traffic, we minimize the distance travelled on the interposer layer and attempt to route packets on the chip network as much as possible. For core-to-core packets, this attempts to mimic the baseline system where the interposer is minimally used.

4.3 Network Topology

In an interposer-based system employing a monolithic multi-core chip such as the one in Figure 2.5, the NoC traffic can be cleanly separated into cache coherence traffic routed on the CPU layer, and main memory traffic on the interposer layer [24]. This approach of differentiating between traffic classes in the network into two physical networks can exhibit numerous benefits [59, 61]. This can help avoid protocol-level deadlock. It also minimizes the interference and contention between the traffic types and more easily allows for per-layer customized topologies that best match the respective traffic patterns. When the multi-core chip has been broken down into smaller pieces, coherence traffic between cores on different chips must now venture off onto the interposer. As a result, this mixes some amount of

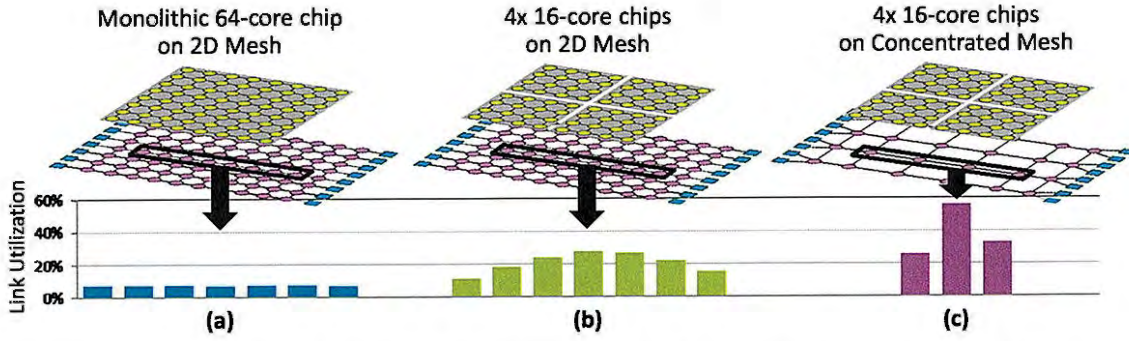


Figure 4.3: Link utilization from a single “horizontal” row of routers on the interposer for (a) a monolithic 64-core stacked on a interposer with a 2D mesh, (b) four 16-core chips stacked on a 2D mesh, and (c) four 16-core chips stacked on a concentrated mesh.

coherence traffic in with the main memory traffic, which in turn disturbs the traffic patterns observed on the interposer. This can be slightly reduced by using minimal interposer routing but it cannot be eliminated entirely.

Figure 4.3 shows per-link traffic for a horizontal set of routers across the interposer for several topologies. The first is the baseline case for a monolithic 64-core chip stacked on an interposer that also implements a 2D mesh. All coherence stays on the CPU die, and memory traffic is routed across the interposer, which results in relatively low and even utilization across all links. Figure 4.3(b) shows four 16-core chips on top of the same interposer mesh. Traffic between chips must now route through the interposer, which is reflected by an increase particularly in the middle link right between the two chips. Figure 4.3(c) shows the same four chips, but now stacked on an interposer with a concentrated-mesh network. Any traffic from the left side chips to the right must cross the middle links, causing further contention with memory traffic. The utilization of the middle link clearly shows how the bisection-crossing links can easily become bottlenecks in multi-chip interposer systems.

In addition to regular and concentrated mesh topologies, shown in Figure 4.4(a) and (b), we consider two additional baseline topologies for the interposer portion of the NoC to address the traffic patterns induced by chip disintegration. The first is the “Double Butterfly” [24] which *optimizes the routing of traffic from the cores to the edges* of the interposer where the memory stacks reside. The Double Butterfly in Figure 4.4(c) has the same number of nodes as the CMesh, but provides the same bisection bandwidth as the conventional mesh. Next, we consider is the Folded Torus², shown in Figure 4.4(d). Similar to the Double Butterfly, the Folded Torus provides twice the bisection bandwidth compared to the CMesh. The Folded Torus actually can provide “faster” east-west transit as each link spans a distance of two routers, but main-memory traffic may not be routed as efficiently as a Double Butterfly due to the lack of the “diagonal” links. Both of these topologies assume the same 4-to-1 concentration as the CMesh.

4.3.1 Misaligned Topologies

When using either Double Butterfly or Folded Torus topologies on the interposer layer, overall network performance improves substantially over either the conventional or concentrated meshes (see Section 5.2). However, the links that cross the bisection between the two halves of the interposer still carry a higher amount of traffic and continue to be a bottleneck for the system. We now introduce the

²This is technically a 2D Folded Torus, but omit the “2D” for brevity.

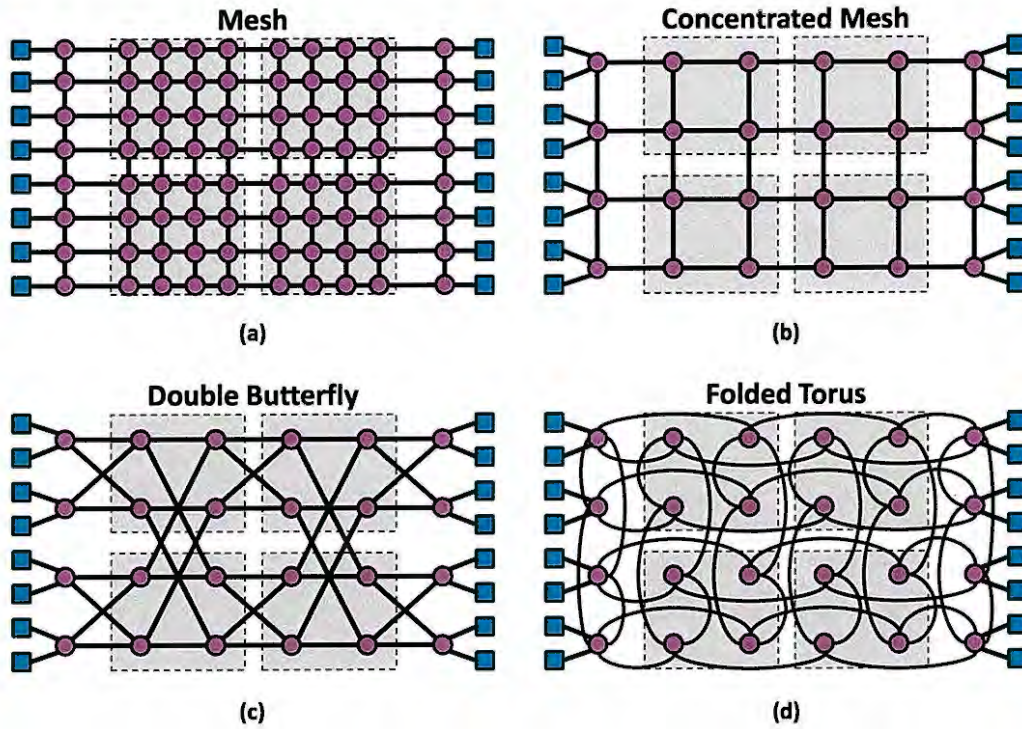


Figure 4.4: Baseline topologies for the interposer portion of the NoC, including (a) 2D Mesh, (b) 2D Concentrated Mesh, (c) Double Butterfly, and (d) 2D Folded Torus. The squares on the edges are memory channels; the four large shaded boxes illustrate the placement of four 16-core chips above the interposer.

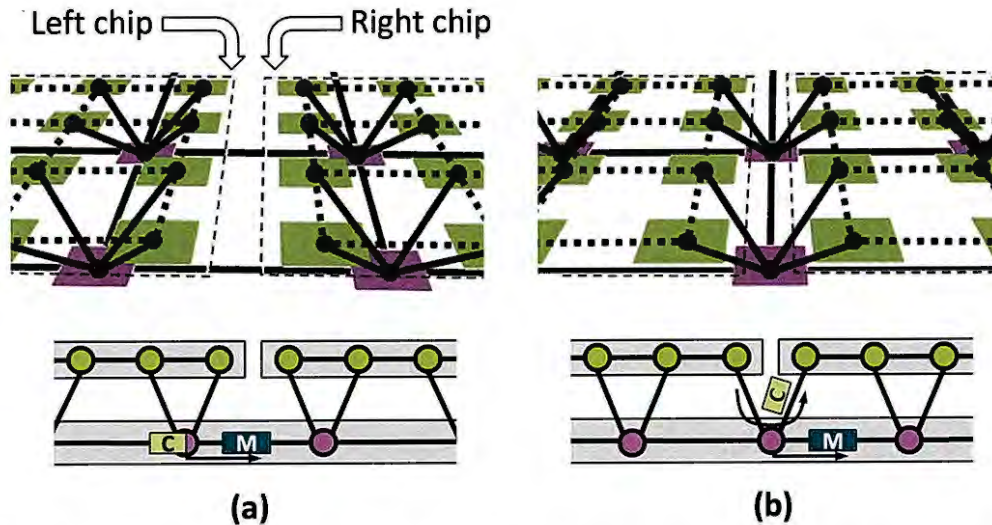


Figure 4.5: Perspective and side/cross-sectional views of (a) 4-to-1 concentration from cores to interposer routers aligned beneath the CPU chips, and (b) 4-to-1 concentration misaligned such that some interposer routers are placed "in between" neighboring CPU chips. The cross-sectional view also illustrates the flow of example coherence (C) and memory (M) messages.

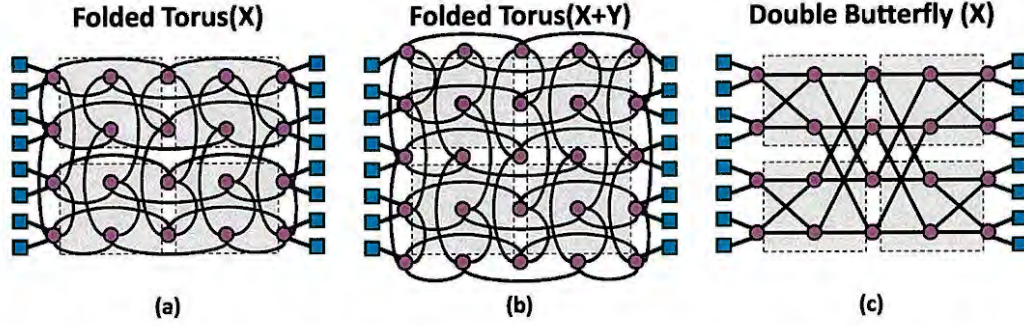


Figure 4.6: Example implementations of misaligned interposer NoC topologies: Folded Torus misaligned in the (a) X-dimension only, (b) both X and Y, and (c) a Double Butterfly misaligned in the X-dimension.

concept of a “misaligned” interposer topology. For our concentrated topologies thus far, every four CPU cores in a 2×2 grid share an interposer router that was placed in between them, as shown in both perspective and side/cross-sectional views in Figure 4.5(a). So for a 4×4 16-core chip, there would be four “concentrating” router nodes aligned directly below each quadrant of the CPU chip.

A misaligned interposer network offsets the location of the interposer routers. Cores on the edge of one chip now share a router with cores on the edge of the adjacent chip as shown in Figure 4.5(b). The change is subtle but important: with an “aligned” interposer NoC, the key resources shared between chip-to-chip coherence and memory traffic are the links crossing the bisection line, as shown in the bottom of Figure 4.5(a). If both a memory-bound message (M) and a core-to-core coherence message (C) wish to traverse the link, then one must wait as it serializes behind the other. With misaligned topologies, the shared resource is now the *router*. As shown in the bottom of Figure 4.5(b), this simple shift allows chip-to-chip and memory traffic to flow through a router simultaneously, thereby reducing queuing delays for messages to traverse the network’s bisection cut.

Depending on the topology, interconnect misalignment can be applied in one or both dimensions. Figure 4.6(a) shows a Folded Torus misaligned in the X-dimension only, whereas Figure 4.6(b) shows a Folded Torus misaligned in both X- and Y-dimensions³. Note that misalignment changes the number of nodes in the topology (one fewer column for both examples, and one extra row for the X+Y case). For the Double Butterfly, we can only apply misalignment in the X-dimension as shown in Figure 4.6(c) because misaligning in the Y-dimension would change the number of rows to five, which is not amenable to a butterfly organization that typically requires a power-of-two in the number of rows.

4.3.2 The ButterDonut Topology

One of the key reasons why both Double Butterfly (DB) and Folded Torus (FT) topologies perform better than the CMesh is that they both provide twice the bisection bandwidth. In the end, providing more bandwidth tends to help both overall network throughput and latency (by reducing congestion-related queuing delays). One straightforward way to provide more bisection bandwidth is to add more links, but if not done carefully, this can cause the routers to need more ports (higher degree), which increases area and power, and can decrease the maximum clock speed of the router. Note that the topologies considered thus far (CMesh, DB, FT) all have a maximum router degree of eight for the

³We do not consider Y-dimension only misalignment as we assume that memory is placed on the east and west sides of the interposer.

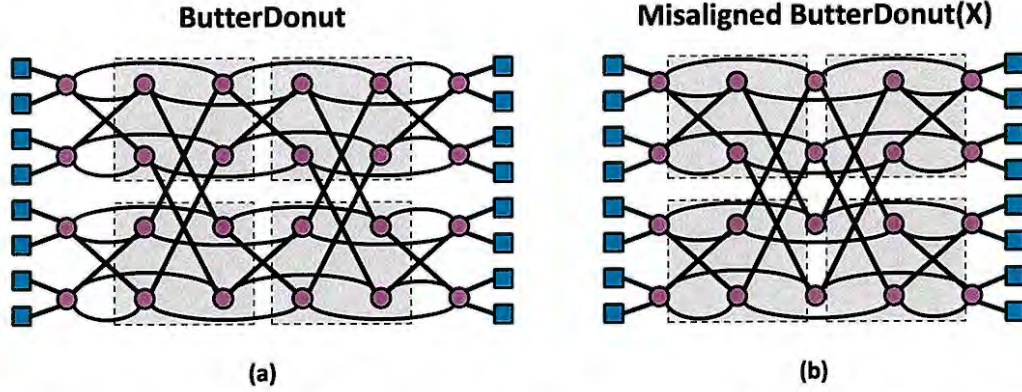


Figure 4.7: Our proposed ButterDonut Topology: (a) aligned and (b) misaligned topologies combine topological elements from both the Double Butterfly and Folded Torus.

interposer-layer routers (i.e., four links concentrating from the cores on the CPU chip(s), and then four links to other routers on the interposer).

By combining different topological aspects of both DB and FT topologies, we can further increase the interposer NoC bisection bandwidth without impacting the router complexity. Figure 4.7 shows our *ButterDonut*⁴ topology, which is a hybrid of both the Double Butterfly and the Folded Torus. All routers have at most four interposer links (in addition to the four links to cores on the CPU layer); this is the same as CMesh, DB, and FT. However, as shown in Figure 4.7(a), the ButterDonut has twelve links crossing the vertical bisection (as opposed to eight each for DB and FT, and four for CMesh).

Similar to the DB and FT topologies, the ButterDonut can also be “misaligned” to provide even higher throughput across the bisection. An example is shown in Figure 4.7(b). Like the DB, the misalignment technique can only be applied in the X-dimension as the ButterDonut still makes use of the Butterfly-like diagonal links that require a power-of-two number of rows.

4.3.3 Comparing Topologies

Topologies can be compared via several different metrics. Table 4.1 shows all of the concentrated topologies considered along with several key network/graph properties. The metrics listed correspond only to the interposer’s portion of the NoC (e.g., nodes on the CPU chips are not included), and the link counts exclude both connections to the CPU cores, as well to the memory channels (this is constant across all configurations, with 64 links for the CPUs and 16 for the memory channels). Misaligned topologies are annotated with their misalignment dimension in parenthesis, for example, the Folded Torus misaligned in the X-dimension is shown as “FoldedTorus(X)”.

As shown in the earlier figures, misalignment can change the number of nodes (routers) in the network. From the perspective of building minimally-active interposers, we want to favour topologies that minimize the number of nodes and links to keep the interposer’s A_{crit} as low as possible. At the same time, we would like to keep network diameter and average hop count low (to minimize expected latencies of requests) while maintaining high bisection bandwidth (for network throughput). Overall, the X-misaligned ButterDonut topology has the best properties out of all of the topologies except for the link count, for which it is a close second behind DoubleButterfly(X). ButterDonut(X) combines the

⁴“Butter” comes from the Butterfly network, and “Donut” is chosen because they are torus-shaped and delicious.

	Topology	Nodes	Links	Diameter	Avg Hop	Bisection Links
Misaligned	CMesh	24 (6x4)	38	8	3.33	4
	DoubleButterfly	24 (6x4)	40	5	2.70	8
	FoldedTorus	24 (6x4)	48	5	2.61	8
	ButterDonut	24 (6x4)	44	4	2.51	12
	FoldedTorus(X)	20 (5x4)	40	4	2.32	8
	DoubleButterfly(X)	20 (5x4)	32	4	2.59	8
	FoldedTorus(XY)	25 (5x5)	50	4	2.50	10
	ButterDonut(X)	20 (5x4)	36	4	2.32	12

Table 4.1: Comparison of the different interposer NoC topologies studied in this paper. In the node column, $n \times m$ in parenthesis indicates the organization of router nodes. Bisection Links are the number of links crossing the vertical bisection cut.

best of all of the other non-ButterDonut topologies, while providing 50% more bisection bandwidth.

4.3.4 Deadlock Freedom

The Folded Torus and ButterDonut topologies are susceptible to network-level deadlock due to the presence of rings within a dimension (either along the X-axis or the Y-axis) of the topology. Two conventional approaches have been widely employed to avoid deadlock in torus networks: virtual channels [2] and bubble flow control [48]. *Virtual channels* (VCs) are separate buffers/queues which share the physical link in a router. An escape virtual channel can be used to ensure deadlock freedom. In an escape VC, a deadlock-free routing function is used (usually by restricting certain turns). When a packet is stuck in the network for a certain period of time (when it is under deadlock), it moves into the escape VC which it can use to reach the destination without another chance of deadlocking. In this work, we leverage recently proposed flit-level bubble flow control [15, 44] to avoid deadlock in these rings. However, with torus-based networks, even by restricting turns, there are rings in a single dimension which has the potential to form a deadlock cycle. *Bubble Flow Control* [48] is a flow control protocol targeted at these types of networks to mitigate this issue and it does so using just a single virtual channel. Puente *et al.* show that for wormhole switching in a torus-based network, each uni-directional ring is deadlock-free if there exists at least one worm-bubble located anywhere in the ring after packet injection. As the ButterDonut topology only has rings in the x-dimension, bubble flow control is applied in that dimension only and typical wormhole is applied for packets transiting through the y-dimension⁵.

4.4 Physical implementation

As discussed in Section 3.2.1, we advocate for a minimally-active interposer. To implement the NoC on an active interposer⁶, we simply place both the NoC links (wires) and the routers (transistors) on

⁵ This discussion treats diagonal links as y-dimension links. For the Folded Torus, bubble flow control must be applied in both dimensions. As strict dimension order routing cannot be used in the ButterDonut topology (packets can change from x to y and from y to x dimensions), an additional virtual channel is required. We modify the original routing algorithm for the DoubleButterfly networks [24]; routes that double-back (head E-W and then W-E on other links) are not possible due to disintegration. Table-based routing based on extended destination tag routing coupled with extra VCs maintain deadlock freedom for these topologies.

⁶Current publicly-known designs have not implemented an active interposer but we believe there is a strong case for it in the future.

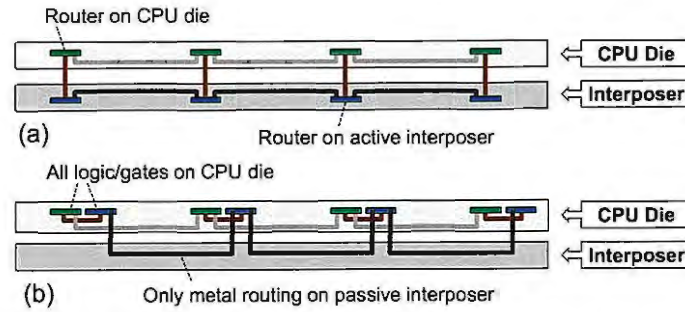


Figure 4.8: (a) Implementation of a NoC with routers on both the CPU die and an active interposer, and (b) an implementation where all routing logic is on the CPU die, and a passive interposer only provides the interconnect wiring for the interposer's portion of the NoC.

the interposer layer. However, most current implementations use a passive interposer. Figure 4.8(a) shows a small example NoC with the interposer layer's partition of the NoC completely implemented on the interposer. For the near future, however, it is expected that only passive, device-less interposers will be commonly used. Figure 4.8(b) shows an implementation where the active components of the router (e.g., buffers, arbiters) are placed on the CPU die, but the NoC links (e.g., 128 bits/direction) still utilize the interposer's routing resources. This approach enables the utilization of the interposer's metal layers for NoC routing at the cost of some area on the CPU die to implement the NoC's logic components.⁷ Both NoCs in Figure 4.8 are topologically and functionally identical, but have different physical organizations to match the capabilities (or lack thereof) of their respective interposers.

4.4.1 μ bump Overheads

We assume a μ bump pitch of $45\mu\text{m}$ [53]. For a 128-bit bi-directional NoC link, we would need 270 signals (128 bits for data and 7 bits of side-band control signals in *each* direction) taking up 0.55mm^2 of area. For an active interposer, each node on the chip requires one set of vertical interconnects. For each 16-core chip, this will take up 8.8mm^2 of area (from the top metal layer) which amounts to less than 15% of the CPU chip area.

For a passive interposer, if the interposer layer is used to implement a mesh of the same size as the CPU-layer, each node would need to have four such links (for each N/S/E/W direction). The total area overhead for the μ bumps for each 16-core chip would be 35.2mm^2 , or nearly half (58%) of our assumed $7.75\text{mm} \times 7.75\text{mm}$ multi-core processor die. To reduce the μ bump area overheads for a passive-interposer implementation, we use concentration [6]. Every four nodes in the CPU layer's basic mesh are concentrated into a single node of the interposer-layer NoC. Figure 4.5 shows different views of this. The side view illustrates the interposer nodes as logically being on the interposer layer (for the passive interposer case, the logic and routing are split between the CPU die and the interposer as described earlier). Usage of a concentrated topology for the interposer layer provides a reduction of the μ bump overheads by a factor of four, down to 8.8mm^2 - the same overhead as with an active interposer design. Concentration for the active interposer case does not have any benefit since each

⁷If the "interposer links" are too long and would otherwise require repeaters, the wires can "resurface" back to the active CPU die to be repeated. This requires some additional area on the CPU die for the repeaters as well as any corresponding μ bump area, but this is not significantly different than long conventional wires that also need to be broken up into multiple repeated segments.

node would still require just one set of vertical interconnects- the same as the base case.

Summary

In this chapter, we described the various designs that we proposed, specifically for 2.5D multi-chip interposer systems. In the next chapter, we explain the methodology we used to evaluate our proposed designs and analyze the results from this methodology.

Chapter 5

Methodology and Evaluation

In this chapter, we first describe the methodology used for the evaluation of the NoC designs from Chapter 4. We then do a extensive evaluation using multiple methods including synthetic workloads as well as full-system simulations. We then present these results and our observations about them.

5.1 Methodology

To comprehensively evaluate our designs, we first tested the various topologies using synthetic traffic on a cycle-level network simulator. We then tested out designs using SynFull [5] workloads. Finally, we looked at Full-System simulations using a cycle-accurate multi-core processor simulator.

5.1.1 Synthetic Workloads

To evaluate the performance of various 2.5D NoC topologies for our disintegrated systems, we use BookSim, a cycle-level network simulator [34], with the configuration parameters listed in Table 5.1. BookSim provides set a basic network topologies, routers, flow control, and arbiters.

Network Topologies and Routing

BookSim does not, by default, support the types of network topologies that we described in Chapter 4. We designed a new method of specifying the various nodes and routers in the system where we can specify the topology of the chip network as well as the interposer network. The interposer topology, in most cases, had to be re-defined for each case as they were unique to our design. We also introduced flags to control whether the interposer topology was to be misaligned or not. The size of the chip can also be varied in terms of the number of nodes along the x-axis and y-axis independently. We also enable the specification of its link latency (which is a default parameter) as well as a link priority. The link priority is used to generate the routing table: this is done using a minimum-path algorithm. By choosing an appropriate set of priorities for each type of link in the system, it is possible to choose between minimal-interposer routing and interposer-first routing. Additionally, we can specify the routing algorithm for each of the two networks.

Common Parameters (all routers)	
VCs	8, 8-flit buffers each
Pipeline	4 stages
Clock	1GHz
Multi-core Die NoC Parameters	
all configs	Standard 2D mesh, DOR routing
Interposer NoC Parameters	
Mesh, Cmesh, All Folded Tori	DOR routing
All DoubleButterfly variants	Extended destination tag routing
ButterDonut variants	Table-based routing

Table 5.1: NoC simulation parameters

Synthetic Traffic Patterns

BookSim provides a set of synthetic traffic patterns which can be used to quickly evaluate various networks topologies. Some example traffic patterns include: uniform random, permutation, and transpose. Based on the traffic pattern, the injection pattern for each node in the system would be different. With uniform random traffic pattern, all nodes will inject to all other nodes at an equal rate. In the case of transpose traffic pattern, each node will only communicate with its transpose node and vice versa.

BookSim synthetic traffic patterns are intended for homogeneous systems. However, in our design, the system consists of two types of nodes: CPU/caches and directories. Directories behave differently from the CPU nodes. CPU nodes both inject and reply to requests sent to it whereas directories do not initiate any communication by themselves. To cater to this, we modified BookSim to use the Request-Reply mode and forced the directories nodes to only reply to requests and not initiate communication. We can then split the traffic to divide it into coherence traffic (core-to-core) and memory traffic (core-to-memory/memory-to-core) in a ratio of our choosing. For the results presented in this thesis (unless otherwise mentioned), the traffic is split 50-50 between coherence and memory traffic. Other ratios did not result in significantly different behaviours.

5.1.2 SynFull Simulations

While synthetic workloads may be able to represent some realistic workloads at some points during their execution, they do not capture the changing behaviours of real-world applications. The advantage that they do have is that the simulation times are very low in comparison to other methods, finishing within minutes. While Full-System simulation is the most representative set of experiments that can be run, they tend to take a large number of compute hours (ranging from a day to over a week for longer workloads). The intermediate solution is to use SynFull [5] Workloads.

SynFull is a synthetic traffic generation methodology which is able to better represent real applications. The SynFull workloads are based on 16-core multi-threaded PARSEC [7] applications. The models that SynFull provides capture the application and the coherence behaviour and allows for rapid evaluation of NoCs (in comparison with Full-System simulations). SynFull is also able to capture the changing behaviour of an application during the course of its execution, including bursty traffic. The

CPU Configuration	
Chip	4, 16-core chips (64 cores in total)
Core	2GHz, Out-of-Order, 8-wide, 192-instruction ROB
L1 Caches	32kB private each for L1-Instruction, L1-Data, 2-way assoc
L2 Caches	512kB shared, unified, distributed cache, 8-way assoc
Coherence Protocol	Directory-Based MOESI
Memory Configuration	
Type	4, 3D-Stacked DRAM
Channels	4-channels per stack
Data Bus	128-bit
Bus Speed	1.6GHz

Table 5.2: Gem5 Configuration for Full-System Simulation

authors of SynFull [5] provide their source code as well as an interface with BookSim for a 16-core system. We use SynFull in a multi-programmed environment (with four instances of SynFull) and interface it with BookSim. We use a non-trivial mapping of each of the SynFull nodes to our network; the four instances are interleaved onto the network so that each 16-core chip has four cores from each instance of SynFull. This prevents the localization of a single program on to a single chip (which would eliminate chip-to-chip communication). Additionally, we include a simple latency-based memory model for the memory controllers.

As mentioned earlier, the authors provide models for a subset of the PARSEC [7] benchmarks. We cluster these benchmarks into one of three categories based on the Per-Node Average Accepted Packet Rate (for the entire simulation). We denote these clusters as *low*, *medium*, and *high* groups. The high group is likely to stress the network more than the medium group.

- **Low:** barnes, blackscholes, bodytrack, cholesky, fluidanimate, lu_cb, raytrace, swaptions, water_nsquared, water_spatial
- **Medium:** facesim, radiosity, volrend
- **High:** fft, lu_ncb, radix

Since each of these models only simulates a 16-threaded model, we combine four models to obtain a multi-programmed workload that makes use of all 64 cores in our system. We interleave the threads from the four programs such that each chip has four threads from each program. To obtain each combination of four programs, we chose four benchmarks at random from the three groups (L=low, M=medium, H=high) to form a number of workloads. We limited the combinations such that each workload falls into one of the following sets: L-L-L-L, M-M-L-L, M-M-M-M, H-H-L-L, H-H-M-M. We compute the geometric mean for each set and present them in Subsection 5.2.1.

5.1.3 Full-System Simulations

Full-System simulators simulate all of the cores, caches, and memory sub-system in addition to the network. For our experiments, we use gem5 [9], a cycle-accurate multi-core processor simulator to simulate the cores, caches, and memory sub-system. We interface gem5 with BookSim [34] and use BookSim as the network simulator. The system configuration is shown in Table 5.2.

We simulate a 64-core SMP system running the Linux kernel version 2.6.28. Each core has its own private L1 cache and a slice of the *shared* L2 cache. There are 16 directories in total. The memory controllers are co-located with the directories. The cores, caches, and directories are mapped to their

corresponding nodes in the network in BookSim. With this setup, we run multi-threaded (with 64 threads) versions of the PARSEC [7] applications (simmedium dataset). The threads of the applications are distributed across the 64 cores and they share/contend for all 16 memory channels. We use a trivial allocation of the threads in this study where threads are allocated from the north-west corner (moving across first, then down). Each program is executed for over one billion instructions.

Simulation Framework

These evaluation approaches cover a wide range of network utilization scenarios and exercise both cache coherence traffic and memory traffic. With the exception of the cost-performance analysis in Section 3.2.1, our performance results do *not* factor in benefits from speed binning individual chips in a disintegrated system. Our performance comparisons across different granularities of disintegration show only the cycle-level trade-offs of the different configurations. If the binning benefits were included, then the overall performance benefits of our proposal would be even greater.

5.2 Experimental Evaluation

In this section, we explore network performance under different chip-size assumptions and compare latency and saturation throughput across a range of aligned and misaligned topologies. Our evaluation uses synthetic workloads, SynFull traffic patterns, and full-system workloads. We considered the link latency for the μ bump to be similar to the link latency of the on-chip network due to the similarity in the impedance of the μ bump and on-chip wiring [29]. We also evaluate the power and area of the proposed topologies and present a unified cost-performance analysis.

5.2.1 Performance

Figure 5.1(a) shows the average packet latency for the Mesh, CMesh, DB, FT, and ButterDonut topologies assuming uniform random traffic with 50% coherence requests and 50% memory requests at a 0.05 injection rate. At a low injection rate, latency is primarily determined by hop count; as expected, the mesh performs the worst. As the number of cores per die decreases, the mesh performance continues to get worse as more and more packets must pay the extra hops on to the interposer to reach their destinations. For the other topologies, performance actually *improves* when going from a monolithic die to disintegrated organizations. This is because the static routing algorithms keep coherence messages on the CPU die; in a monolithic chip, the coherence messages miss out on the concentrated, low hop-count interposer networks. DB, FT, and ButterDonut all have similar performance due to similar hop counts and bisection bandwidths.

At low loads, the results are similar for other coherence/memory ratios as the bisection links are not yet a bottleneck. The results show that any of these networks are probably good enough to support a disintegrated chip at low traffic loads. Disintegrating a 64-core CPU into four chips provides the best performance, although further reduction in the sizes of individual chips (e.g., to 8 or even 4 cores) does not cause a significant increase in average packet latency.

Figure 5.1(b) shows the average packet latency for Folded Torus (FT), Double Butterfly (DB), and ButterDonut along with their misaligned variants. Note that the y-axis is scaled up to make it easier

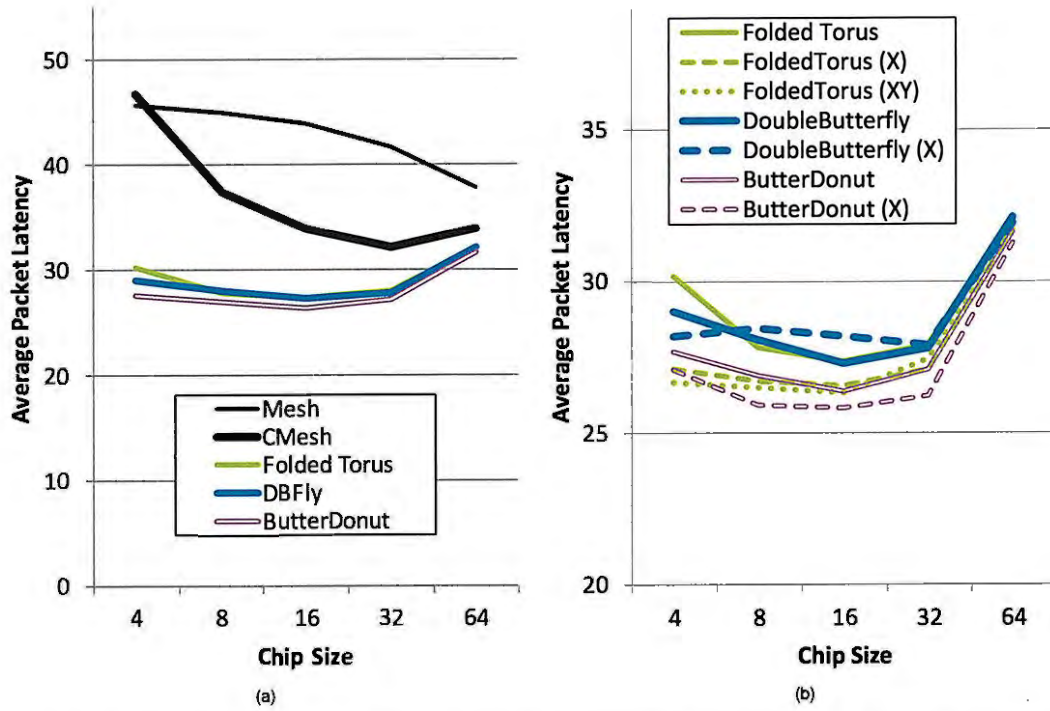


Figure 5.1: Average packet latency for different interposer NoC topologies. The x-axis specifies the individual chip size (16 = four 16-core chips, 64 = a single monolithic 64-core chip). Results are grouped by (a) aligned and (b) misaligned topologies. Traffic is split 50-50 between coherence and memory, with a 0.05 injection rate.

to see the differences among the curves. The misaligned topologies generally reduce network diameter/hop count, and this is reflected by the lower latencies. The Folded Torus enjoys the greatest reductions in latency from misalignment. For the aligned and misaligned topologies, ButterDonut and ButterDonut(X) respectively have the lowest average packet latency for low injection synthetic workloads.

Figure 5.2 shows average packet latency results when the system executes multi-programmed SynFull workloads. The results are for a system consisting of four 16-core CPU chips. The SynFull results show a greater difference in the performance between the different topologies as the workloads exercise a greater diversity and less uniform set of routes. Across the workloads, the misaligned ButterDonut(X) and FT(XY) consistently perform the best. The mesh typically performs the worst since it does not have the long links that the concentrated networks benefit from. We also observe that misalignment generally improves the performance of the network. In some cases, FT(XY) loses performance over FT(X) and FoldedTorus. Misalignment in the Y-axis would yield benefits for the coherence traffic. However, since the number of directories is still restricted to eight (with four routers) on each side, misaligning in the Y-axis adds an extra hop for all nodes in the top and bottom rows of the interposer layer.

Figure 5.3 shows histograms for several interposer topologies' packet latencies for a system with four 16-core chips running uniform random traffic at a 0.05 injection rate. The CMesh suffers from both the highest average and highest variance in packet latency. The other higher-bandwidth solutions all have similarly low and tight latency distributions, with the ButterDonut performing the best. The

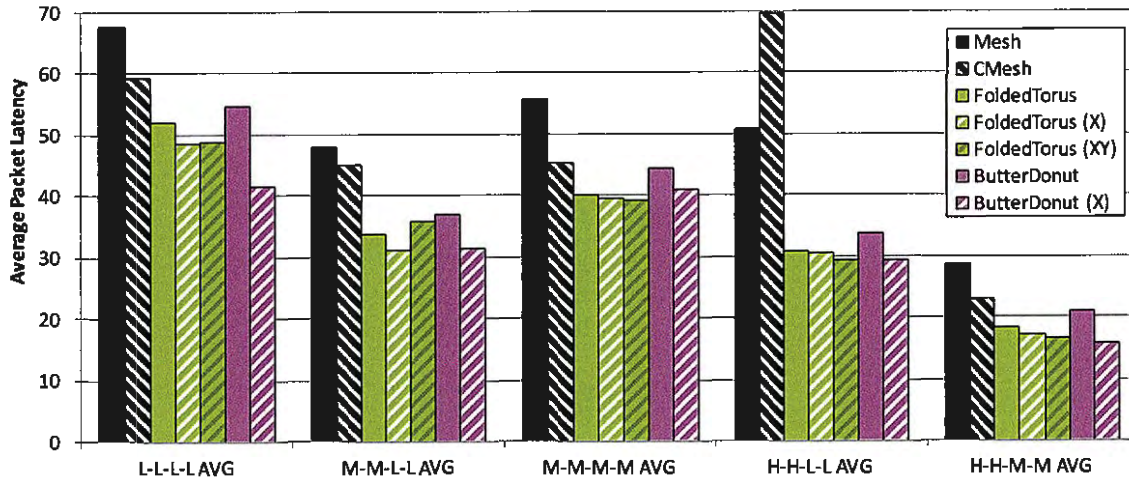


Figure 5.2: Average packet latency results for different multi-programmed SynFull workloads, with each benchmark denoted by the combination of the groups they are derived from (based on average accepted injection rate)

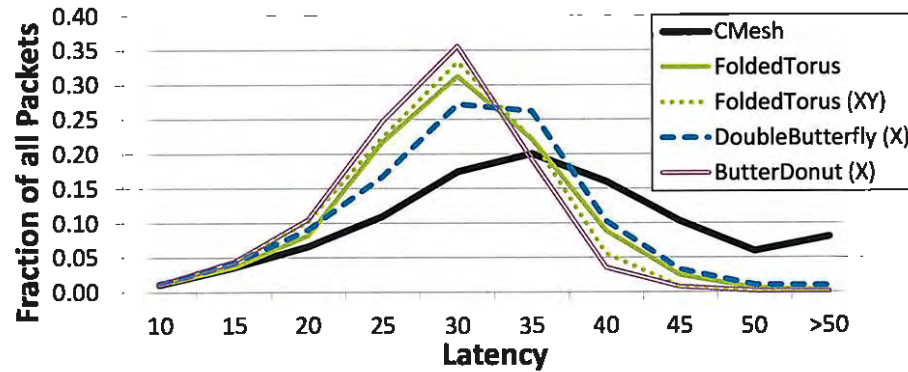


Figure 5.3: Distribution of message latencies (0.05 injection rate)

low average hop counts of these NoCs keep average latency down, and the higher bandwidth reduces pathological traffic jams that would otherwise result in longer tails in the distributions.

Figure 5.4 shows the full-system results for several PARSEC [7] benchmarks (with the simmedium dataset) normalized to a mesh interposer network. These results are consistent with the network latency results in Figure 5.1. For workloads with limited network or memory pressure, many topologies exhibit similar network latencies which results in little full-system performance variation across topologies.

The primary characteristics of each of the benchmarks we evaluated is shown in Table 5.3. Blacksholes has a small working set and little data sharing or exchange. The various threads, once they begin, are largely independent of one another. Due to this, blacksholes puts limited pressure on the memory system and the network and therefore sees little difference across topologies. Bodytrack is also a data-parallel model and puts a little more pressure on the network. We observe a little more variation across the topologies. We also notice a slightly larger improvement over the baseline (interposer mesh network). Canneal has an unstructured parallel-programming model. It performs cache-aware simulated annealing. An important aspect of canneal is that it makes use of an aggressive synchronization strategy. It promotes data race recovery instead of avoidance [8]. As such, it makes very little use of

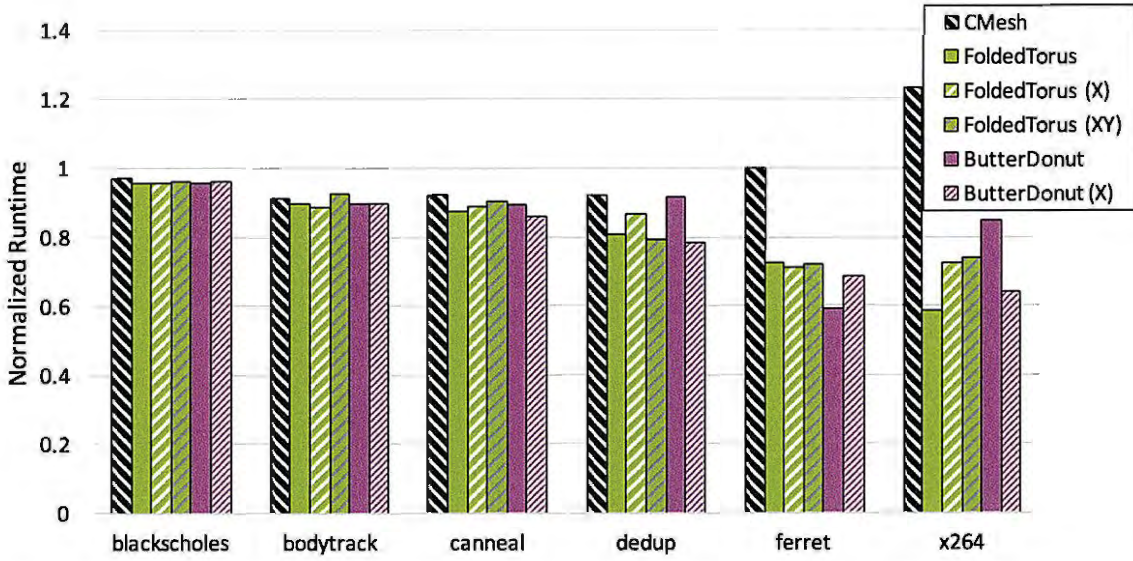


Figure 5.4: Normalized runtime (To Mesh Interposer Network) for full-system simulation with a four 16-core chips

Application	Application Domain	Parallelization		Working Set	Data Usage	
		Model	Granularity		Sharing	Exchange
blackscholes	Financial Analysis	data-parallel	coarse	small	low	low
bodytrack	Computer Vision	data-parallel	medium	medium	high	medium
canneal	Engineering	unstructured	fine	unbounded	high	high
dedup	Enterprise Storage	pipeline	medium	unbounded	high	high
ferret	Similarity Search	pipeline	medium	unbounded	high	high
x264	Media Processing	pipeline	coarse	medium	high	high

Table 5.3: Characteristics of the PARSEC benchmarks used in this study [8]

locks or barriers. Thus, we see around 10% improvement over the baseline when using concentration but little variability across the different topology. Dedup, ferret, and x264 are more memory intensive workloads. These kernels have a pipelined programming model to parallelize the application. They also have high data sharing and exchange and a large number of locks and waits (on condition variables) [8]. Due to this, we see large improvements with the FoldedTorus and the ButterDonut. We also see that misalignment improves the system performance in many cases. In some cases, we notice a performance drop (which we did not observe with synthetic workloads or SynFull workloads). This is likely due to the way we allocated the threads for the full-system applications. Since we started allocating threads from the north-west corner, the master thread inevitably gets allocated to the corner node. Misalignment, on general, improves performance but has a negative effect on some nodes (including the nodes on the edge, especially the ones on the corner). We believe that we would see improved performance with a random or a smart thread-to-core allocation mechanism.

5.2.2 Load vs. Latency Analysis

The SynFull simulations hint at the impact of network load on the performance of the NoC. Because SynFull is designed to mimic the actual injection rates and patterns of real applications, it is difficult

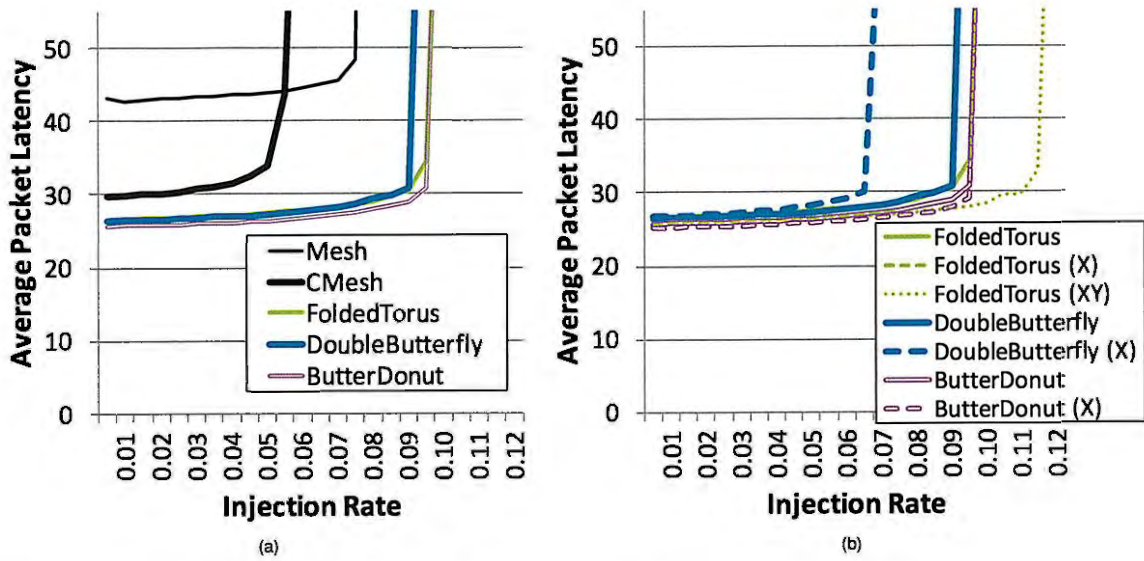


Figure 5.5: Latency and saturation throughput for (a) conventional NoC topologies and (b) misaligned topologies.

to use it to arbitrarily sweep a range of traffic loads. To generate latency-load plots, we revert back to synthetic uniform random traffic (50-50 coherence and memory). We evaluate a system with four 16-core chips. Figure 5.5(a) and (b) show the latency-load curves for the same topologies shown earlier in Figure 5.1. For the aligned topologies, the CMesh quickly saturates as its four links crossing the bisection of the chip quickly become a bottleneck. The remaining topologies maintain consistently low latencies up until they hit saturation. The misaligned topologies generally follow the same trends, although FT(XY) shows a substantially higher saturation bandwidth. The reason is that the Butterfly-derived topologies are optimized for east-west traffic to route memory traffic, and as a result are slightly imbalanced. Looking back at Figure 4.7, ButterDonut has 12 links across the east-west bisection, but only eight links going across the north-south cut. FT(XY) on the other hand has ten links across both bisections, and scales out better. The cost (see Table 4.1) is that FT(XY) has 25% more router nodes and 39% more links than ButterDonut, which can impact the yield of the minimally-active interposer.

Figure 5.6 further breaks down the load-latency curves into (a) memory traffic only, and (b) cache coherence traffic only. The results are fairly similar across the two traffic types, with the memory traffic having a slightly larger spread in average latencies due to the higher diversity in path lengths (memory requests always have to travel farther to the edges). The plots come from the same set of experiments, and therefore the saturation onset occurs at the same point in both graphs.

In general, from standard topologies to their misaligned variants, latency decreases and saturation throughput improves. Saturation throughput improvements come from taking pressure off of the east-west bisection links in the aligned topologies. Latency reductions come from shorter hop counts in these networks.

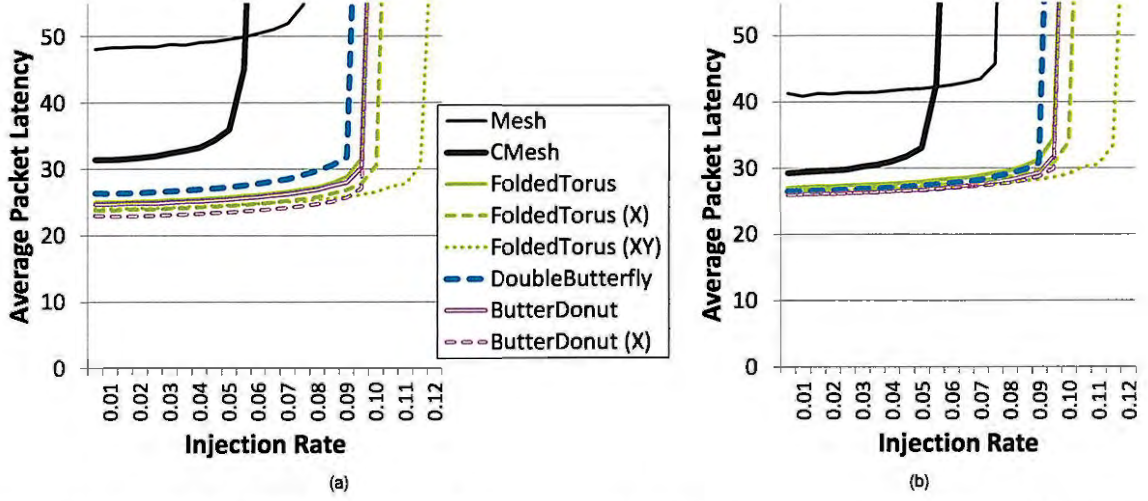


Figure 5.6: Latency and saturation throughput separated into (a) only memory messages, and (b) only core-to-core coherence messages.

5.2.3 Routing Protocols

We discussed two types of routing protocols to control the flow of traffic between the interposer network and the on-chip network in Section 4.2. Due to the long computation time of Full-System and SynFull workloads, we evaluated the two protocols using BookSim with synthetic traffic (50-50 coherence and memory).

Figure 5.7(a) and (b) show the side-by-side results for interposer first routing and minimal interposer routing. We can observe that with minimal interposer routing, the saturation point for various topologies are higher than those with interposer first routing. However, before saturation, the average packet latency is typically lower for interposer first routing. With minimal-interposer routing, the coherence traffic resides on chip and uses the interposer only for edge-to-edge routing, leading to the higher saturation point. However, with minimal-interposer routing, since the coherence traffic does not make use of the low-latency, long interposer links, the average network latency is higher. On the other hand, with interposer first routing, the interposer network is more stressed since it caters to $3/4$ of the coherence traffic and all of the memory traffic. Thus, it tends to reach saturation a little earlier. Since interposer first routing provides a better performing network on average, we used interposer first routing for most of our experiments.

5.2.4 Power and Area Modelling

We use DSENT [55] as described below to estimate frequency, area, and power for the different topologies for the entire network (both the on chip network and the interposer network). We assume all NoC topologies are clocked at 1GHz. All links, routers (including input buffers, ports, crossbar), and router placement are modelled faithfully. Long links needed by some topologies are properly repeated (in DSENT) in order to meet timing. If the interposer is built using an older process technology, there would be variations in the capacitance (of the wires) per unit length. However, since the process nodes that we are assuming (22nm vs 45nm) for the interposer and the chips are relatively close as compared to other cases (such as Virtex-7 [50]), the variation would not be significant. We are also running the

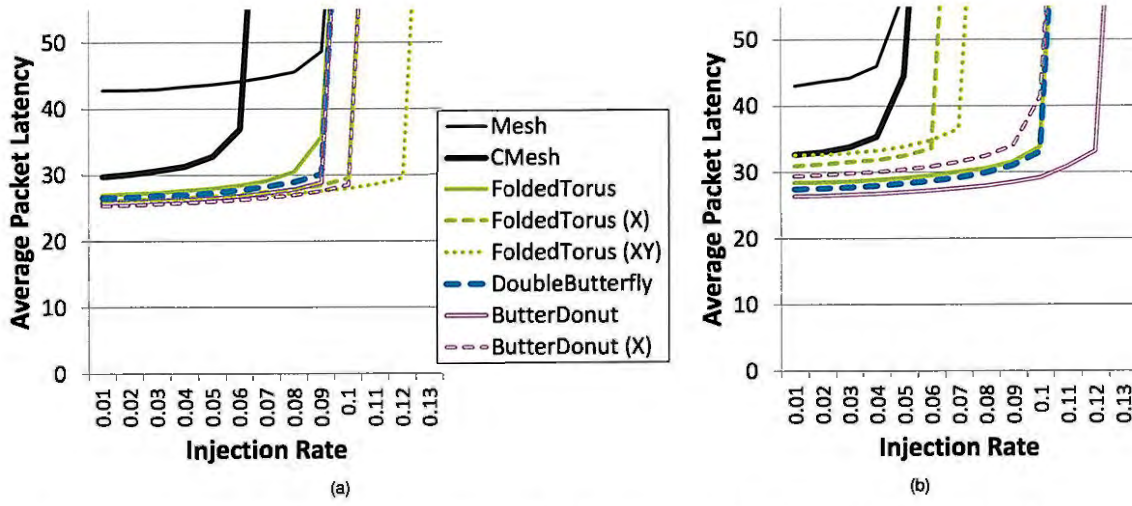


Figure 5.7: Routing Protocols (a) Interposer First Routing, and (b) Minimal Interposer Routing

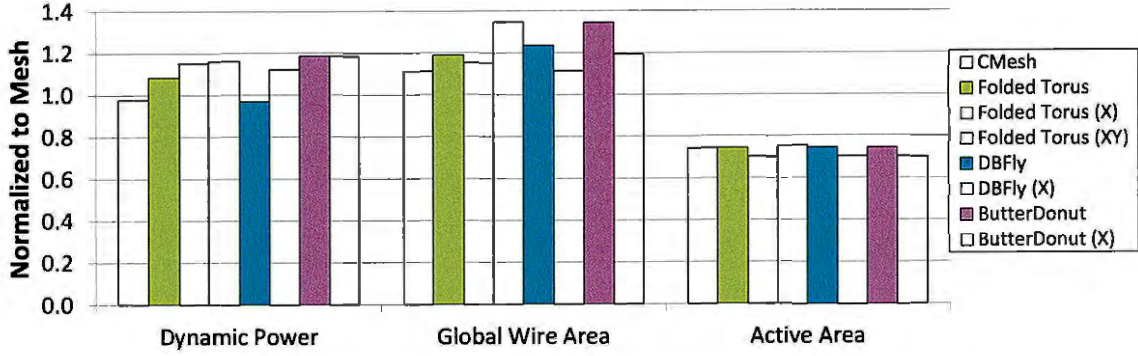


Figure 5.8: Power and area results in 45nm normalized to mesh

running at a relatively slower speed (at 1GHz) which is well supported by both of the process nodes.

All of our configurations assume an appropriately sized mesh on the core dies (e.g., a 16-core chip uses a local 4×4 mesh). All interposer networks use 4-to-1 concentration except for the mesh. There are two DRAM stacks on each of the two sides of the interposer. Each DRAM stack provides four independent memory channels, for a system-wide total of 16 channels. Each DRAM channel consists of a 128-bit data bus operating at 1.6GHz. DRAM-specific timing (e.g., bank conflicts, refresh) are modelled. The interposer network dimensions include the end-nodes that interface with the DRAM memory channels. Results are collected using a 45nm bulk/SOI low- V_t process node with 3.4mm^2 cores, with a worst-case CPU-layer core-to-core link length of 2.0mm (1.6mm CPU width, plus 0.4mm for μ bump area). The frequency was swept to determine the maximum operating point for each topology. Long link lengths are faithfully modelled across each topology. μ bump power was computed using the activity factors measured from simulation, a system voltage of 0.9V, and a μ bump capacitance of 0.7fF [29]. We also check if we are able to meeting the timing requirements using DSENT.

In Figure 5.8, we show power and area results normalized to a mesh for each topology. This considers only the network power and area but includes the network on each of the chips, the interposer network, and the μ bumps. We show three metrics: dynamic power (total dynamic power consumed by

Peak Operating Frequency	Topologies
2.5 GHz	ButterDonut*, FT(X), FT(XY)
2.75 GHz	DBFly*
3.0 GHz	Mesh, Cmesh*, FT

Table 5.4: Peak Network Operating Frequency as Reported by DSENT. “*” represents a topology and all of its misalignments.

the network), global wire area (the total wiring area used on all four chips and the interposer), and active area (area required to implement all the routers). Longer links for some topologies (e.g., FT(XY), DB and ButterDonut) lead to increased global wiring area; however increases are minor and do not exhaust the routing budget of the interposer. Likewise, longer links will consume more dynamic power (due to the presence of more repeaters); however the absolute power consumed by these topologies is small and will have a negligible impact on the overall thermals of the system. We obtain clock frequencies for all of the topologies. DSENT reports reduced clock frequency for all topologies with longer links. This is shown in Table 5.4. ButterDonut has the lowest operating frequency due to long links. However, it can still be clocked at 2.5GHz which is more than sufficient to saturate the DRAM bandwidth. The power results assume 1GHz frequency for all topologies.

Using the network power results (from DSENT) and the full system results, we can look at the overall change in energy utilization during a run of an application. We observe 1% improvement in the network energy for ButterDonut and 7% improvement for ButterDonut(X).

5.2.5 Combining Cost and Performance

Higher levels of disintegration (smaller chips) result in lower overall cost based on our analysis from Section 3.1.1 due to better yields and more chips per wafer. The smaller chips decrease interconnect performance due to fragmentation of the NoC, but the finer-grained binning also increases average CPU clock speeds. To put it all together, we consider two figures of merit (FOM) based on cost and performance, or “delay”. The delay/performance metric that we use is the average network latency adjusted by the average clock-speed improvement due to binning (Monte Carlo simulations were repeated for all chips sizes considered).

For performance, we assume that only 30% of performance is impacted by the CPU frequency while the NoC frequency remains unchanged and that only 20% of the system performance is affected by the NoC performance (in terms of the average packet latency for each network). With these simplified assumptions, we compute the performance improvement due to CPU frequency (from our speed binning results in Section 3.1.1) and due to gains from the NoC using average packet latency improvements over the baseline (with uniform random synthetic traffic divided into 50% coherence traffic and 50% memory traffic). The cost metric is the cost per 64-core system (considering yield and chips per wafer). As we are only comparing interposer-based systems, the cost of the interposer is not factored into the cost metric. Both delay and cost are normalized against the monolithic 64-core.

The first, $FOM^{\times}$, is the product of delay and cost, which gives both factors similar influence. The second, $FOM^{\wedge}$, is $delay^{cost}$, which provides a greater emphasis on performance. The rationale for the performance-heavy FOM is that for high-end servers, even relatively small performance differentiations at the high-end can translate into substantially higher selling prices and margins.

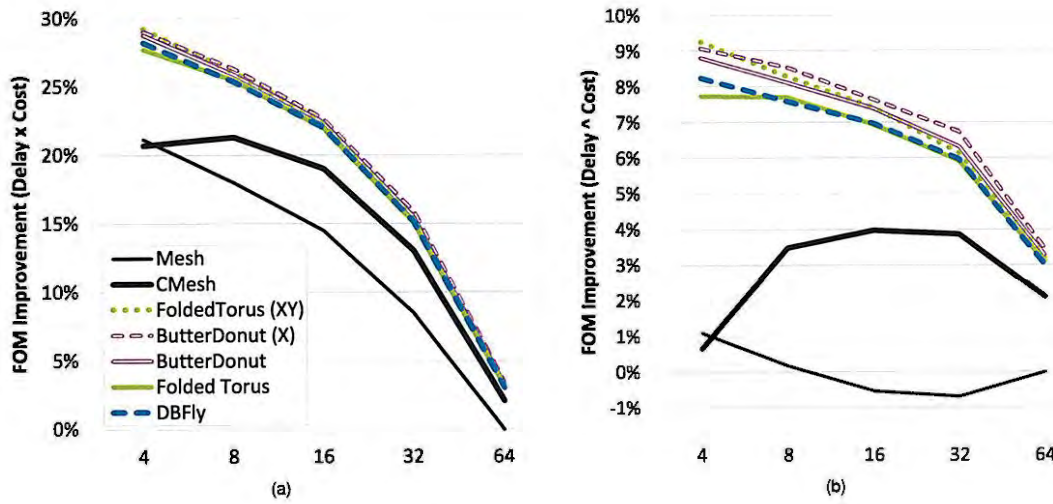


Figure 5.9: Figures of Merit based on (a) delay $\times$ cost and (b) delay^{cost}. The x-axis specifies the number of core per chip (64=monolithic).

Figure 5.9(a) and (b) show $FOM^{\times}$ and $FOM^{\wedge}$, respectively. For the cost-delay product ($FOM^{\times}$), even though higher levels of disintegration cause average NoC performance to degrade, these are more than offset by the cost benefits combined with CPU frequency improvements. The exponential $FOM^{\wedge}$ tells a more interesting story. For a basic Mesh on the interposer, the performance loss due to disintegration actually hurts more than the cost reductions help until one gets down to using 8-core chips or smaller. The CMesh provides an initial benefit with two 32-core chips, but the lower bisection bandwidth of the CMesh is too much of a drag on performance, and further disintegration is unhelpful. The $FOM^{\wedge}$ results show that the remaining topologies are effectively able to ward off the performance degradations of the fragmented NoC sufficiently well that the combination of binning-based improvements and continued cost reductions allow more aggressive levels of disintegration.

With different FOMs, the exact trade-off points will shift, but $FOM^{\wedge}$ in particular illustrates that simple disintegration (using Mesh, CMesh) alone may not be sufficient to provide a compelling solution for both cost and performance. However, interposer-based disintegration appears promising when coupled with an appropriate redesign of the interposer NoC topology.

5.2.6 Clocking Across Chips

The NoCs span multiple die and the interposer. Thus far we have modelled a fully-synchronous NoC where all nodes on all chips are clocked together. Considering die-to-interposer and die-to-die parametric variations and the possibility of implementing the interposer and individual chips in different technology generations, trying to build a high-speed, low-skew, low-jitter global clock tree across multiple chips and the interposer is likely to be *very* challenging. While it makes sense to run each chip's portion of the NoC at the same clock speed (otherwise the slowest portion will likely become a bottleneck), from a physical-design perspective it is easier to clock chips independently. Thus, each chip's NoC portion operates in its own local timing domain, thereby requiring domain-crossing synchronizing FIFOs when going from CPU chip to interposer, and vice versa. If all domains operate at roughly the same frequency, the domain-crossing FIFOs may be relatively low latency (consisting of phase-

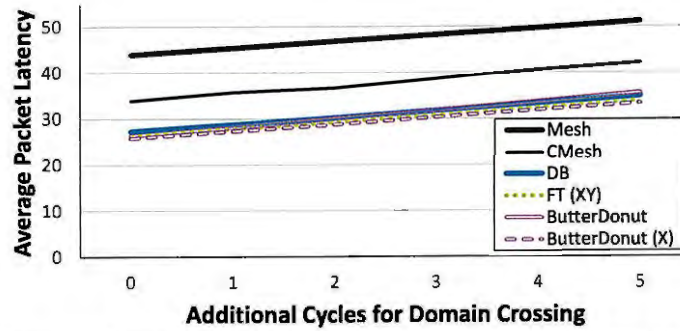


Figure 5.10: Impact of clock crossing latency on average packet latency with 16-core die

detection logic and some registers). If each portion runs at arbitrary clock speeds, then higher-latency synchronizers may be needed.

We repeated some of our experiments where each crossing from a CPU chip to the interposer, and back, must incur an additional 1 to 5 cycles for the FIFOs to synchronize between clock domains. Figure 5.10 shows the average latency for uniform random traffic on a four 16-core configuration. The additional latency affects all of the topologies in a uniform manner, effectively offsetting the performance in proportion to the penalty of crossing timing domains. This shifts the trade-off point in terms of how aggressively one can/should disintegrate a large chip. The higher degree of disintegration (smaller chips) increases the amount of traffic that goes chip-to-chip. Referring back to Figure 5.1, going from a monolithic chip to four 16-core chips decreases average packet latency by about 5 cycles. From Figure 5.10, this would suggest that a domain-crossing FIFO latency of 3-4 cycles would be tolerable to maintain the same performance levels as the original monolithic chip.

5.2.7 Non-disintegrated Interposer NoCs

The topologies that we proposed in this thesis are intended for use with a disintegrated system. We tested the effects of using them on a monolithic chip with 3D-DRAM memory stacks on the interposer. Figure 5.11 shows the results of this study. For the memory traffic in Figure 5.11(a), the results show

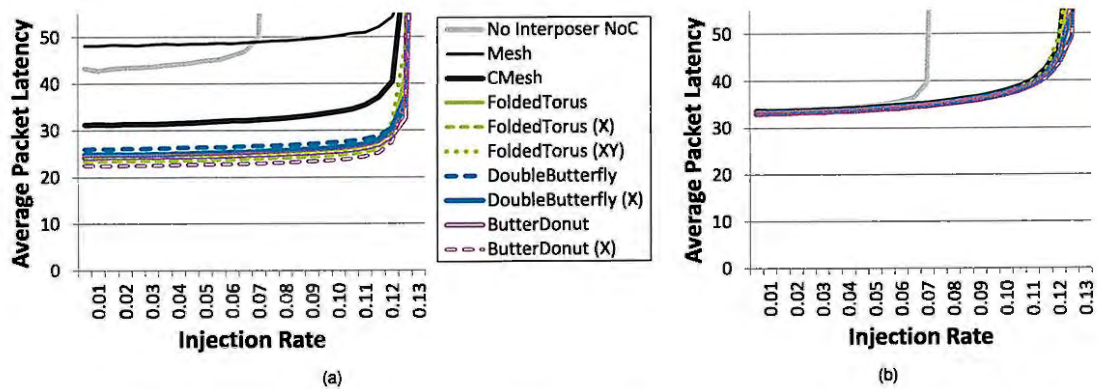


Figure 5.11: Load-latency curves for a monolithic 64-core chip stacked on the interposer, with traffic separated into (a) memory-only and (b) coherence-only traffic.

that our new approach of misaligned topologies improves upon the prior topologies even for a non-disintegrated system. Our best performing topology, ButterDonut(X), provides a $\sim 9\%$ improvement in average packet latency over the best prior topology (the aligned DB). Optimizing NoC designs for an interposer-based *monolithic* SoC was not the goal of our work, but the additional improvements are a welcome result.

Summary

In this chapter, we described the methodology we used to evaluate our designs in terms of power, area, frequency, and performance. We used DSENT for power, area, and frequency analysis. For power, we evaluated our designs using three different methods; we used synthetic traffic workloads, SynFull workloads, and full-system simulations. Between these three types of workloads, we cover a diverse set of test-cases. We include an analysis, comparing two different types of routing protocols and study the effects of clock jitter between the different clocking domains of the interposer network and the on-chip network. We also compared our multi-chip design to a monolithic design. Effectively, we show the benefits of our proposed designs as well as highlight a few of the disadvantages. In the next chapter, we qualitatively compare our proposed design with other recent works.

Chapter 6

Related Work

In this chapter, we discuss related works in the areas of multi-die systems, hierarchical networks, 3D NoC designs, and multi-NoC designs.

Brick and Mortar [40] integrate heterogeneous components on an individual die connected through an interposer using a highly-reconfigurable network substrate that can accommodate different router architectures and topologies [39]. This flexibility leads to a high A_{crit} which negatively impacts interposer yield. Our system, with less heterogeneity, performs well with a regular topology which reduces areas to improve yield and cost.

Through interposer-based disintegration, we have essentially constructed a set of hierarchical topologies (locally within each die, and globally on the interposer). Hierarchical NoCs have been explored for *single-chip* systems. To improve latency for local references, bus-based local networks have been employed [19, 56] with different strategies for global connections. Buses and rings [4] are effective for a small number of nodes; although we use a mesh, alternative intra-die topologies could further reduce complexity and improve latency.

Our 2.5D NoCs bear some similarity to 3D die stacked NoCs [38, 42, 47, 62]. These NoCs are typically identical across all layers and assume arbitrary or full connectivity within a layer. Our system does not allow arbitrary connectivity within the core layer as it consists of multiple die. Considering the system constraints for the hybrid memory cube, Kim *et al.* route all traffic through a memory-centric network leading to longer core-to-core latencies [36]; similarly, we route some core-to-core traffic through the interposer but develop optimized topologies to offset negative performance impacts.

Recent work proposes leveraging multiple networks to improve performance. Multiple physical networks can obviate the need to use virtual channels to break protocol-level deadlock; it has been shown that multiple physical networks is a more affordable solution than multiple virtual networks [61]. Tilera [59] has also taken this approach and separated traffic associated with coherence, memory traffic, I/O, etc. NOC-Out [43] targets specific Cloud workloads and assumes minimal core-to-core communication; their system is highly optimized for core-to-L3-cache (for Instruction cache miss) performance. They use a butterfly to reach L3 slices and a different topology for the cores. Similarly, we use a Double Butterfly to optimize memory traffic, but the traffic patterns and layout of our system are different resulting in a new topology.

When using multiple networks to improve performance [6, 21, 24, 43, 58, 61, 59], traffic can be partitioned across these networks uniformly or functionally. We take a functional partitioning approach.

However, the presence of the interposer makes a large difference: memory traffic uses the interposer, while core traffic stays on the local core mesh. Additionally, due to the presence of multiple separate die for our cores, our partitioning is not strict (some core traffic *must* use the interposer network to reach remote cores).

Chapter 7

Future Directions & Conclusions

7.1 Future Directions

The presence of two networks (on-chip and interposer) allows for a variety of designs that we have not explored in this work. For the on-chip network, we used a standard mesh network. Using different types of networks (including those with concentration) could be a good way to improve coherence traffic on chip. Multiple networks are often used to improve performance. We could potentially use multiple networks on the interposer to improve the performance: we could use prioritization schemes such as [20] or [52] or we could separate different types of traffic at the physical layer (more affordable than using multiple virtual networks [61]).

7.1.1 New Chip-design Concerns

This work assumes the disintegration of a multi-core chip into smaller die, with the implicit assumption that all die are identical. This creates a new physical design and layout challenge as each die must implement a symmetric interface; for example, a given die could be placed on the left side of the interposer or the right, and that same exact die must correctly interface with the interposer regardless of its mounting position. Conventional SoCs have no such requirements for their layouts. This extends beyond functional interfaces, to issues such as power delivery (the power supply must be reliable independent of chip placement) and thermal management (temperature may vary depending on chip location). Many of these challenges are more in the domain of physical design and EDA/CAD tool optimization, but we mention it here to paint a fair picture of some of the other challenges for interposer-based disintegration.

7.1.2 Software-based Mitigation of Multi-Chip Interposer Effects

Disintegration introduces additional challenges and opportunities for optimization at the software level. With a monolithic 64-core NoC, there is already non-uniformity in latency between various source-destination pairs as hop count increases, which is exacerbated in a disintegrated system. Careful scheduling to place communicating threads on the same chips could reduce chip-to-chip traffic and alleviate congestion on the interposer. Similarly, careful data-allocation to place frequently used data in the 3D stacks close to the threads that use the data can minimize memory latency and cut down on

cross-interposer traffic. Coordinating scheduling decisions to simultaneously optimize for both coherence and memory traffic could provide further benefits but would require support from the operating system. While we believe there are many interesting opportunities to involve the software layers to help mitigate the impacts of a multi-chip NoC organization, we leave it for future research.

7.2 Conclusion

In this work, we looked at a conventional baseline 2.5D multi-core system. We made the key insight that if the chip is already using a silicon interposer for integration of multiple chips, any additional resources used on the interposer (provided that timing is met) are essentially “free”. We can then use these resources to either add more computational logic or routing. These resources can be used to design new network topologies as well as integrate multiple chips on a single interposer die.

We propose disintegrating a large monolithic chip into multiple smaller chips. We then integrate the smaller chips on an interposer to replicate the functionality of the larger chip. We do an extensive cost versus performance study of this approach, considering the various trade-offs including area, power, yield, network latency, system performance, and chip speeds. We then design and evaluate NoC topologies for 2.5D architectures. One of our key contributions here is the introduction of “misaligned” concentrated network topologies. These topologies specifically cater to cross-chip communication and show good promise for multi-chip systems. We demonstrate that our designs can significantly improve performance over the baseline mesh network. We show that even when under peak use in our system, the active area (router logic) can be reduced around 30% when using concentration while reaping performance benefits. There is an increase in the global wire area (by 10-30%) as well as increase in dynamic power but the overall energy can be reduced by 7%. It is important to note that this extra wiring comes from the interposer metal layers which was previously unused. Our proposal is able to offer a method to integrate large chips while maintaining (and in cases, improving) performance. By doing so, we may be able to manufacture larger chips than possible on a monolithic design.

Ultimately, we show that instead of using a monolithic chip on a 2.5D interposer system, it is favourable (in terms of energy, area, and performance) to disintegrate the large chip into multiple smaller chips and use an underlying NoC on the interposer to replicate the original function.

Bibliography

- [1] AMD Corp. High Bandwidth Memory (HBM). <http://www.amd.com/Documents/High-Bandwidth-Memory-HBM.pdf>.
- [2] William Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.
- [3] Krit Athikulwongse, Ashutosh Chakraborty, Jae-Sok Yang, David Z. Pan, and Sung Kyu Lim. Stress-Driven 3D-IC Placement with TSV Keep-Out Zone and Regularity Study. In *Intl. Conf. on Computer-Aided Design*, pages 669–674, San Jose, CA, November 2010.
- [4] Rachata Ausavarungnirun, Chris Fallin, Xiangyao Yu, Kevin Change, Greg Nazario, Reetuparna Das, Gabriel Loh, and Onur Mutlu. Design and evaluation of hierarchical rings with deflection routing. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing*, 2014.
- [5] Mario Badr and Natalie Enright Jerger. SynFull: Synthetic traffic models capturing a full range of cache coherence behaviour. In *Intl. Symp. on Computer Architecture*, June 2014.
- [6] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Intl. Conf. on Supercomputing*, 2006.
- [7] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.
- [8] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.
- [9] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. gem5: A Multiple-ISA Full System Simulator with Detailed Memory Model. *Computer Architecture News*, 39, June 2011.
- [10] Bryan Black. Die Stacking is Happening. In *Intl. Symp. on Microarchitecture*, Davis, CA, December 2013.
- [11] Eric Bogatin, Dick Potter, and Laura Peters. *Roadmaps of packaging technology*. Integrated Circuit Engineering Scottsdale, 1997.
- [12] Mark Bohr. A 30 year retrospective on dennard’s mosfet scaling paper. *Solid-State Circuits Society Newsletter, IEEE*, 12(1):11–13, 2007.
- [13] Mark T Bohr et al. Interconnect scaling-the real limiter to high performance ulsi. In *International Electron Devices Meeting*, pages 241–244. INSTITUTE OF ELECTRICAL & ELECTRONIC ENGINEERS, INC (IEEE), 1995.
- [14] Shekhar Borkar. Design challenges of technology scaling. *Micro, IEEE*, 19(4):23–29, 1999.

- [15] Lizhong Chen and Timothy M. Pinkston. Worm-bubble flow control. In *19th Intl. Symp. on High Performance Computer Architecture*, 2013.
- [16] Chung-Kuan Cheng. Breaking the Wall of Interconnect: Research and Education. <http://cseweb.ucsd.edu/~kuan/talk/interconnect.ppt>.
- [17] Jason Cong and Yan Zhang. Thermal-Driven Multilevel Routing for 3-D ICs. In *10th Asia South Pacific Design Automation Conference*, Shanghai, China, January 2005.
- [18] William J Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pages 684–689. IEEE, 2001.
- [19] Reetuparna Das, Soumya Eachempati, Asit K. Mishra, Vijaykrishnan Narayanan, and Chita R. Das. Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps. In *Intl. Symp. on High Performance Computer Architecture*, 2009.
- [20] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R Das. Aéria: exploiting packet latency slack in on-chip networks. In *ACM SIGARCH computer architecture news*, volume 38, pages 106–116. ACM, 2010.
- [21] Reetuparna Das, Satish Narayanasamy, Sudhir K. Satpathy, and Ronald Dreslinski. Catnap: Energy proportional multiple network-on-chip. In *Intl. Symp. on Computer Architecture*, 2013.
- [22] Yangdong Deng and Wojciech Maly. Interconnect Characteristics of 2.5-D System Integration Scheme. In *Intl. Symp. on Physical Design*, pages 171–175, Sonoma County, CA, April 2001.
- [23] Robert H Dennard, VL Rideout, E Bassous, and AR Leblanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.
- [24] Natalie Enright Jerger, Ajaykumar Kannan, Zimo Li, and Gabriel H. Loh. NoC architectures for silicon interposer systems. In *47th Intl. Symp. on Microarchitecture*, pages 458–470, Cambridge, UK, December 2014.
- [25] Michael Hackerott. Die Per Wafer Calculator. Informatic Solutions, LLC, 2011.
- [26] Jen-Hsun Huang. NVidia GPU Technology Conference: Keynote. Technical report, March 2013.
- [27] Ron Huemoeller. Through Silicon Via (TSV) Product Technology. Technical report, Amkor Technology, February 2012. Presented to IMAPS North Carolina Chapter.
- [28] Wei-Lun Hung, Greg Link, Yuan Xie, Narayanan Vijaykrishnan, and Mary Jane Irwin. Interconnect and Thermal-aware Floorplanning for 3D Microprocessors. In *7th Intl. Symp. on Quality Electronic Design*, San Jose, CA, March 2006.
- [29] Institute of Microelectronics. Process Design Kit (PDF) for 2.5D Through Silicon Interposer (TSI) Design Enablement & 2.5D TSI Cost Modeling, August 2012.
- [30] An Intel. Introduction to the intel quickpath interconnect. *White Paper*, 2009.
- [31] Michael Jackson. A Silicon Interposer-based 2.5D-IC Design Flow, Going 3D by Evolution Rather than by Revolution. Technical report, Synopsis Insight Newsletter, 2012. Issue 1.
- [32] JEDEC. High Bandwidth Memory (HBM) DRAM. <http://www.jedec.org/standards-documents/docs/jesd235>.
- [33] JEDEC. Wide I/O Single Data Rate (Wide I/O SDR). <http://www.jedec.org/standards-documents/docs/jesd229>.
- [34] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, and William J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *Intl. Symp. on Performance Analysis of Systems and Software*, 2013.

- [35] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H. Loh. Enabling interposer-based disintegration of multi-core processors. In *48th Intl. Symp. on Microarchitecture*, Hawaii, US, December 2015.
- [36] G. Kim, J. Kim, J-H. Ahn, and J. Kim. Memory-centric system interconnect design with hybrid memory cubes. In *Intl. Conf. on Parallel Architectures and Compilation Techniques*, 2013.
- [37] J-S. Kim, C. Oh, H. Lee, D. Lee, H-R. Hwang, S. Hwang, B. Na, J. Moon, J-G. Kim, H. Park, J-W. Ryu, K. Park, S-K. Kang, S-Y. Kim, H. Kim, J-M. Bang, H. Cho, M. Jang, C. Han, J-B. Lee, K. Kyung, J-S. Choi, and Y-H. Jun. A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking. In *ISSCC*, 2011.
- [38] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Reetuparna Das, Yuan Xie, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures. In *34th Intl. Symp. on Computer Architecture*, San Diego, CA, June 2007.
- [39] M. Mercaldi Kim, J. D. Davis, M. Oskin, and T. Austin. Polymorphic on-chip networks. In *Intl. Symp. on Computer Architecture*, 2008.
- [40] M. Mercaldi Kim, M. Mehrara, M. Oskin, and T. Austin. Architectural implications of brick and mortar silicon manufacturing. In *Intl. Symp. on Computer Architecture*, 2007.
- [41] Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Hongjung Kim, Ju Young Kim, Young Jun Park, Jae Hwan Kim, Dae Suk Kim, Heat Bit Park, Jin Wook Shin, et al. 25.2 a 1.2 v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 432–433. IEEE, 2014.
- [42] Feihui Li, Chrysostomos Nicopoulos, Thomas Richardson, Yuan Xie, Vijaykrishnan Narayanan, and Mahmut Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *33rd Intl. Symp. on Computer Architecture*, pages 130–141, Boston, MA, June 2006.
- [43] Pejman Lotfi-Kamran, Boris Grot, and Babak Falsafi. NOC-Out: Microarchitecting a Scale-Out Processor. In *Intl. Symp. on Microarchitecture*, pages 177–187, Vancouver, BC, December 2012.
- [44] Sheng Ma, Zhiying Wang, Zonglin Liu, and Natalie Enright Jerger. Leaving one slot empty: Flit bubble flow control for torus cache-coherent NoCs. *IEEE Transactions on Computers*, 64:763–777, March 2015.
- [45] Mike O'Connor. Highlights of the High-Bandwidth Memory (HBM) Standard. In *Memory Forum Workshop*, June 2014.
- [46] C. Okoro, M. Gonzalez, B. Vandeveld, B. Swinnen, G. Eneman, S. Stoukatch, E. Beyne, and D. Vandeputte. Analysis of the Induced Stresses in Silicon During Thermocompression Cu-Cu Bonding of Cu-Through-Vias in 3D-SIC Architecture. In *the Electronic Components and Technology Conference*, pages 249–255, Reno, NV, May 2007.
- [47] Dongkook Park, Soumya Eachempati, Reetuparna Das, Asit K. Mishra, Yuan Xie, Narayana Vijaykrishnan, and Chita R. Das. MIRA: A Multi-layered On-chip Interconnect Router Architecture. In *Intl. Symp. on Computer Architecture*, pages 251–261, Beijing, China, June 2008.
- [48] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. M. Prellezo. The adaptive bubble router. *Journal of Parallel and Distributed Computing*, 64(9):1180–1208, 2001.
- [49] Kiran Puttaswamy and Gabriel H. Loh. Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor. In *ACM Great Lakes Symp. on VLSI*, pages 19–24, Philadelphia, PA, May 2006.
- [50] Kirk Saban. Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency. White paper, Xilinx, 2011. WP380 (v1.1).

- [51] T Sakurai and K Tamaru. Simple formulas for two- and three-dimensional capacitances. *IEEE Transactions on Electron Devices*, 30(2):183–185, 1983.
- [52] Joshua San Miguel and Natalie Enright Jerger. Data criticality in network-on-chip design. 2015.
- [53] Mike Santarini. Stacked & Loaded: Xilinx SSI, 28-Gbps I/O Yield Amazing FPGAs. Technical report, Xilinx Xcell Journal, First Quarter 2011.
- [54] C. H. Stapper. The Effects of Wafer to Wafer Defect Density Variations on Integrated Circuit Defect and Fault Distributions. *IBM Journal of Research and Development*, 29:87–97, January 1985.
- [55] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *NOCS*, May 2012.
- [56] Aniruddha N. Udupi, Naveen Muralimanohar, and Rajeev Balasubramonian. Towards scalable, energy-efficient, bus-based on-chip networks. In *Intl. Symp. on High Performance Computer Architecture*, 2010.
- [57] University of Texas. 65nm Wire Capacitance and Resistance Calculator. http://users.ece.utexas.edu/~mcdermot/vlsi-2/Wire_Capacitance_and_Resistance_65nm.xls.
- [58] Stavros Volos, Ciprian Seiculescu, Boris Grot, Naser Khosro Pour, Babak Falsafi, and Giovanni De Micheli. CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache Coherent Servers. In *6th NOCS*, pages 67–74, Lyngby, Denmark, May 2012.
- [59] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-chip interconnection architecture of the Tile Processor. *Micro, IEEE*, 27(5):15–31, Sept.-Oct. 2007.
- [60] Yuan Xie, Gabriel H. Loh, Bryan Black, and Kerry Bernstein. Design Space Exploration for 3D Architecture. *ACM Journal of Emerging Technologies in Computer Systems*, 2(2):65–103, April 2006.
- [61] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni. Virtual channels vs. multiple physical networks: a comparative analysis. In *Design Automation Conference*, 2010.
- [62] Aamir Zia, Sachhindh Kannan, Garret Rose, and H. Jonathan Chao. Highly-scalable 3D Clos NoC for Many-core CMPs. In *NEWCAS Conference*, pages 229–232, Montreal, Canada, June 2010.