Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8887589, 11 pages
https://doi.org/10.1155/2020/8887589

Research Article

Management of Load-Balancing Data Stream in Interposer-Based Network-on-Chip Using Specific Virtual Channels

Mona Soleymani,[1] **Midia Reshadi**,[1] **and Ahmad Khademzadeh**[2]

[1]*Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran*
[2]*Education and International Scientific Corporation Department, Iran Telecommunication Research Center, Tehran, Iran*

Correspondence should be addressed to Midia Reshadi; midia.reshadi@gmail.com

Received 11 April 2020; Revised 8 July 2020; Accepted 27 July 2020; Published 25 August 2020

Academic Editor: Farman Ullah

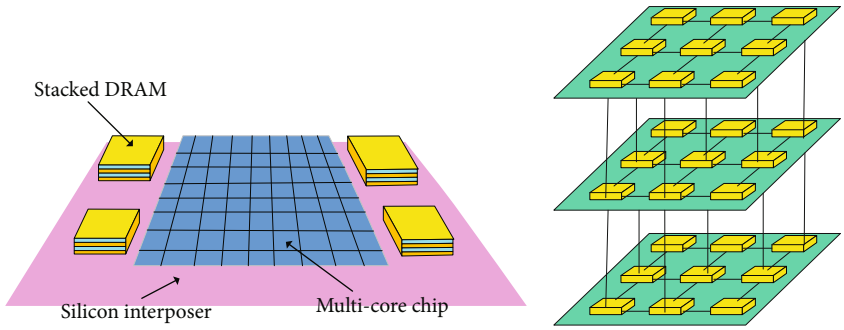
The interaction between cores and memory blocks, in multiprocessor chips and smart systems, has always been a concern as it affects network latency, memory capacity, and power consumption. A new 2.5-dimensional architecture has been introduced in which the communication between the processing elements and the memory blocks is provided through a layer called the interposer. If the core wants to connect to another, it uses the top layer, and if it wants to interact with the memory blocks, it uses the interposer layer. In a case that coherence traffic at the processing layer increases to the extent that congestion occurs, a part of this traffic may be transferred to the interposer network under a mechanism called load balancing. When coherence traffic is moved to the interposer layer, as an alternative way, this may interfere with memory traffic. This paper introduces a mechanism in which the aforementioned interference may be avoided by defining two different virtual channels and using multiple links which specifically determines which memory block is going to be accessed. Our method is based on the destination address to recognize which channel and link should be selected while using the interposer layer. The simulation results show that the proposed mechanism has improved by 32% and 14% latency compared to the traditional load-balancing and unbalanced mechanisms, respectively.

1. Introduction

With the silicon interposer-based technology, a new architecture called 2.5D has been introduced in which an increasing number of memory blocks may be merged throughout multiprocessor chips [1]. What spotlights this architecture is that it increases the amount of memory, which is horizontally located around the processing chip. Unlike the 3D NoC, which the capacity of memory in a package is banned by the size of a processor chip [1–4], in 2.5D technology, the size of the interposer layer determines how much memory blocks are able to be integrated [1]. Figure 1 shows a 2.5D stacking technology with four DRAM stacks on the interposer. Placing on the two horizontal sides, more memory stacks are integrated through the silicon interposer area. For this reason, higher capacities and higher bandwidth are achievable compared to 3D technology [5–7]. In this figure, the differences between 2.5D and 3D technologies have been shown.

In 2.5D architecture, there are two layers: the conventional processing layer, including processing elements, routers, and links, which is located on top layer, and a newly emerged layer called interposer placed below it. The connection between processing cores are regularly established on the top layer; in the case that a core wants to interact with a memory block, it may use the interposer layer to send/receive its packet [8].

In Figure 1(a), it may be seen that memory blocks are located on the silicon interposer and connected with the interposer layer through interface nodes. There are also two disparate types of traffic: the first one is the core-to-core traffic associated with the interaction between cores and the second is the core-to-memory traffic, which is responsible



(a) 2.5D architecture (b) 3D architecture

Figure 1: 2.5D technology vs. 3D technology.

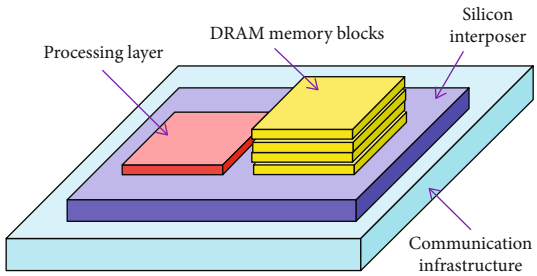


Figure 2: 3D view of 2.5D network-on-chip.

for transferring packets between memory blocks and cores [1, 6]. Core-to-core traffic, which is known for coherence traffic, should be distributed across all the networks in order to avoid protocol-level deadlock [1]. Each of aforementioned types of traffic has its own characteristics [1]. Coherence traffic is established based on peer to peer communication patterns while core-to-memory traffic generates a many-to-few traffic pattern [1]. As each of these traffic patterns is utilized for different purposes, they should be segregated from each other. Figure 2 shows another view of the 2.5D network-on-chip that memory blocks are just presented on the right side of the processing layer.

One of the characteristics of the interposer layer is the support of different heterogeneous integrations and structures [9–11].

As it may be seen in Figure 3, disparate components with different functions may be integrated through the interposer layer. Internet of Things may be a favorable heterogeneous utilization in which multifunctional and compact devices with high performance and low energy consumption are needed. Figure 4 demonstrates two disparate traffics in two different paths. In a conventional 2D network-on-chip, a processing core may send its packet to another processing core using blue routers; similarly, a processing core may interact with a memory block through green routers. The internal structure of a tile, containing an element and a router alongside cache memories, is also presented in Figure 4.

Considering the upper layer, when the volume of core-to-core coherence traffic is exceedingly increasing, congestion is potentially created due to heavy workload while the lower layer seems to be underutilized [1]. As a result, the performance will be weak [8]. When the congestion occurred in the processing layer, nodes cannot potentially send/receive their packets easily and there are lots of time to be wasted as the routers. This will spoil the performance of routing and switching since some packets are stuck to some routers and there is no way to go ahead. Latency is the most valuable parameter which is negatively affected due to this phenomenon. At this time, a mechanism called load balancing has been proposed to alleviate the congestion and related problems [8].

What the load-balancing mechanism does is to balance loads of traffic on the existing network layers and enhance the utilization of available resources [8]. With regard to the volume of traffic between memory-core and intercore communication, it seems logical that the number of packets streamed across the cores is higher than the memory traffic. This means that the data from one processing node to another similar one flowed more frequently than the request of reaching memory blocks [1]. This is the reason why congestion is supposed to be seen in the processing layer while the interposer layer does hardly face congestion, if not at all. In order to recognize whether the load-balancing method should be utilized or not, the volume of the coherence traffic at the top layer is supposed to size up. Some methods have been proposed to achieve the congestion information which may be categorized into two different groups: buffer-aware and latency-aware. In the former method, according to the filling capacity of each neighbor router, the information of congestion may be detected, which is not globally reliable because of the locality [12]. So, the buffer-aware strategy is not appropriate in 2.5D NoC. In a study proposed by Chen et al. [8], known as a dynamic latency-aware load balancing (DLL), the latency of congestion packets may be properly recorded. This is mainly because this information is tracked by the clock in every router until it arrives at the destination nodes. We use the DLL congestion detection method to recognize when the load-balancing mechanism should be used according to the congestion data; this strategy is designed based on latency-aware architecture.

In the load-balancing method, after detecting congestion in the upper layer, it is possible to send some packets randomly to the interposer layer so as to move and reach their destinations. Actually, they use the interposer layer as an alternative way to escape from the heavy workload they are confronted with at the top layer. As mentioned earlier, the

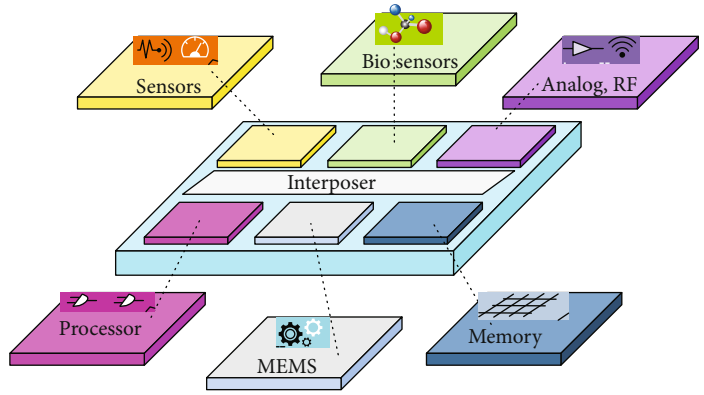


Figure 3: The interposer layer which provides different integrations with various technologies on a single system.

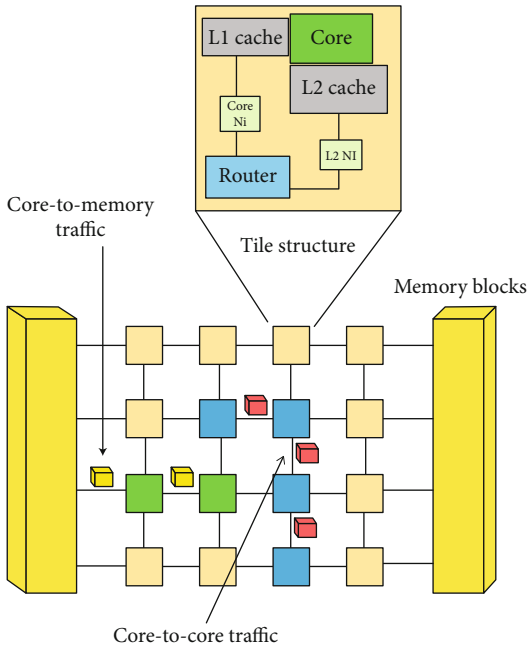


Figure 4: Different types of traffic.

interposer layer is targeted to use for core-to-memory traffic, so by sending the core-to-core traffic to the lower layer, these two types of packets are susceptible to interfere. In this paper, we aim to propose a way in which the load-balancing mechanism does not result in conflict. This happens due to the separation of traffics by utilizing virtual channels and multiple links.

First, we want to explain that the communication between the different components is managed differently in our work. Core-to-core communication is different from core-memory communication. Communication between core-to-core occurs via routers. The router is provided with an entry in the form of a kernel message or whatever adjacent components. The router is partitioned into two layers, i.e., the packet layer and the circuit layer. By the type and the size of the data to be transferred via the routers, the router layer is used. A packet-switched NoC is used to transfer small amounts of data and in a synchronized communication, whereas to transfer data transfers such as multimedia and big data applications, the circuit-switched router (NoC) is used, which is flexible and configures all combinatorial cable connections, resulting in less latency and better energy efficiency. The core-memory communication is done via the memory pipeline access ports dedicated by a wired connection. All memory is shared and accessible by pipeline from all cores within clusters.

The rest of the paper is arranged as follows: Section 2 tends to represent related work. In section 3, we attempt to assume a target structure and show how our proposed method may effectively manage and control the load-balancing mechanism. Experimental results are drawn in Section 4, and finally, Section 5 presents the conclusion.

1.1. Related Work. In order to take advantage of 3D die stacking, several papers have been recently published [13–16]. However, traffic has not been differentiated in many of the proposed methods [1], which means traffic from any types may be passed through each of the layers. Xu et al. [17] proposed that long links may be utilized to customize every 3D layer and this also depletes the hop counts for all traffics. Disparate physical layers have been proposed in [18] in which virtual channels are used to categorize coherence traffic types to hinder protocol-level deadlock. Furthermore, 3D NoC architecture faces some issues, namely, thermal and cooling troubles, the lack of EDA tools, and many problems related to the test [19]; for this reason, 2.5D NoC, which is based on silicon stacking, has recently been popular [5, 20]. That is, many design tools [1, 6, 7, 21] support 2.5D stacking architecture and likewise, many use it for further GPU designs [1, 22]. It is worth mentioning that this newly emerged technology is already seen in many commercial products [1, 23, 24]. Jerger et al. and Li et al. were the pioneers to conduct research and investigate the effect of the interposer in 2.5D NoC in favor of space [1, 8]. Their work first focused on how different topologies may affect the load-balancing strategy in 2.5D interposer-based architecture and then illustrated the destination latency-aware method as an effective way of detecting information related to the congestion [1]. In the aforementioned work, packets are sent through the paths and received by other paths, and as a result, an unsustainable congestion detection method was used for network selection [8]. Figure 5 shows an example of an interleaved-based system,

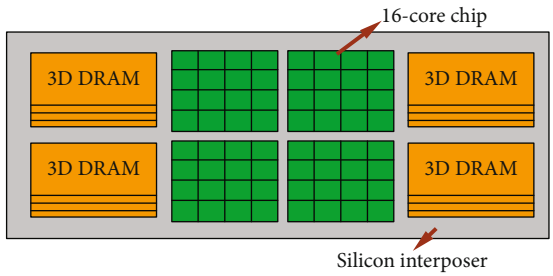


Figure 5: 64-core system composed with DRAM [25].

including four 16-core processing possibilities, an interconnect replica, and a quartet of HBM DRAM stacks located along the left and right sides of the processors.

For recognizing the area related to the congestion, another suitable strategy called Dynamic Latency-aware Load-balancing (DLL) mechanism [8] was proposed. Like Enright's work, the DLL method has used the latency-aware strategy, and it is implemented into different stages: detecting and collecting congestion data and distributing it towards source nodes. Apart from topologies evaluated in [1], other topologies in 2.5D network-on-chip have been proposed such as [25]. The topology named ClusCross used in [25] defines any small chip as a cluster and utilizes long links to easily have access to memory blocks, which boost the bisection bandwidth and deplete average hop count.

Other works like [26] has addressed some issues which results in avoiding bottleneck by looking into the new interposer design space of passive and active interposer technologies, its topologies, and clocking schemes to determine the cost-optimal interposer architectures. The work in [27] has proposed a method called EIRs (Equivalent Injection Routers) which transforms the few-to-many traffic pattern to many-to-many pattern along with the interposer links. EIR scheme solves the bottleneck problem as well as enhancing throughput among manycore processors.

The load-balancing method has been introduced by Jerger et al. to be used in the 2.5D interposer network-on-chip, but this is not controlled and managed. In our previous paper, we have introduced a limited method called CLBM (Controlled Load-Balancing Mechanism) which controls the load-balancing method by defining a forbidden area [28]. Furthermore, in the aforementioned paper, a multicast ring has been introduced to propagate the latency packet across the source nodes. Although in our previous work we have controlled the load-balancing mechanism, it yet does not separate various types of traffic and just prevents the edge grids from being the hotspot. In this paper, we try to use the interposer layer as a second option, when congestion is detected in the processing layer, through a manageable and controllable way that ensures there is no conflict and interference between coherence and core-memory traffic.

2. Proposed Method

In this section, we introduce our 2.5D target structure and the conventional load-balancing strategy. Then, the potential challenges in relation with the load-balancing strategy on interposer-based NoC are explored and we explain how using virtual channels may manage and control traffic interference.

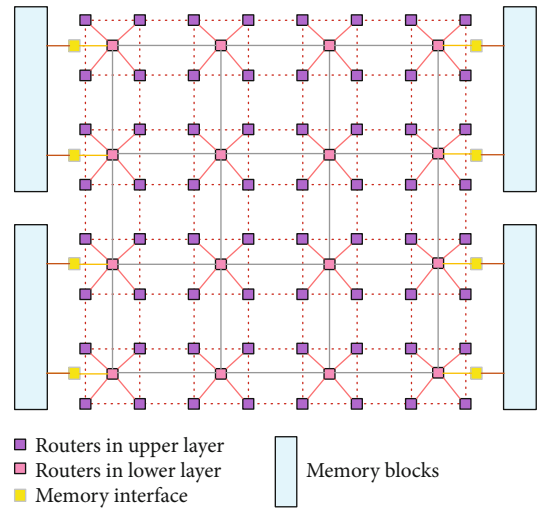


Figure 6: The view of the processing layer and the interposer layer topology.

2.1. Target Structure. Our target structure, in this paper, is based on a 64-core processing unit and 4 stacked memory blocks integrated on the 2.5D network-on-chip architecture. Our processing layer is designed by the mesh topology, containing all the processing cores, while the lower layer embodies the interposer routers which are responsible for memory interaction. TSVs (Through Silicon Via) and μbumps connect these two layers. Because μbumps take a heavy toll on the network [29], the concentrated mesh has been proposed to be used for the interposer routers and this decreases the count of routers. According to the concentrated mesh, every four processing cores are attached to an interposer router. The memory blocks are placed on the horizontal sides of the interposer layer.

Figure 6 illustrates all the routers in a view. There are some yellow nodes known as memory interface nodes, which make a connection between edge interposer routers and the memory blocks.

As mentioned earlier, we may divide the traffic into two different categories: the first one is related to the interaction between processing cores with each other and we know them as CtC. The other is about the connection between processing cores and memory blocks known as CtM in this paper.

2.2. Load-Balancing Mechanism. As usual, in the interposer-based network-on-chip, CtC packets go through the CPU layer and the interposer layer is utilized for CtM traffic [1]. The volume of CtC traffic sometimes appears to be exceeding from a specific threshold, whereas a less number of cores have a request for interacting with memory [1, 8]. In such a scenario, congestion is going to happen on the processing network, meaning that the workload of CtC is more than the CtM traffic. Detecting congestion in the CPU layer is synonymous with the use of the load-balancing mechanism. In

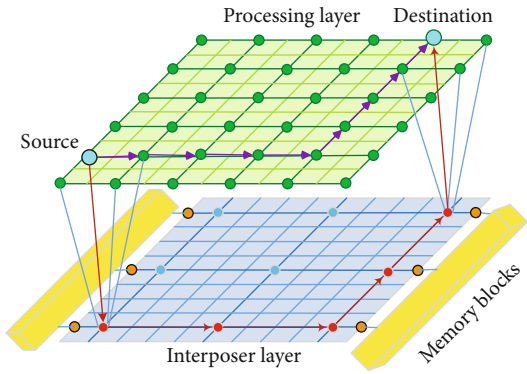


Figure 7: An illustration of load balancing.

this method, some CtC-related packets may choose the interposer layer to move and route. When they approach the nearest node of their destination, they tend to be transferred to the upper layer again and reach their destination node. Figure 7 illustrates how the load-balancing mechanism works. According to this figure, source and destination nodes want to communicate with each other and both of them as a part of CtC traffic are located on the processing layer. Conventionally, all CtC packets are supposed to be routed on the CPU layer in purple links through eight hops. Given the fact that congestion has occurred in the upper layer, the source node sends its packet to the interposer router attached to it. This packet is routed in the interposer layer and after crossing six hops back to the top layer and reach its destination. Even though the length of links has increased, the number of hops has reduced.

We empirically consider 10 cycles as a basic threshold to recognize congestion for interlayer networks. There are still other criteria established when traffic is permitted to move across the second network [30]. The nodes in the interposer layer have been called grids. In the load-balancing mechanism, the CtC traffic in the upper layer is passed through the interposer layer and this will interfere with CtM which is basically associated with the interposer layer. This interference not only may disturb the process of having access to memory blocks for CtM but also may create undesirable and mixed congestion in the lower layer. This paper is aimed at segregating these two types of traffic when a load-balancing mechanism is used.

2.3. Proposed Strategy. This paper is aimed at separating CtC traffic from CtM ones when both try to use the interposer layer as a network to reach their destination. Our strategy is implemented on the routers associated with the interposer layer so as to recognize how to send each individual packet. As mentioned earlier, routers in the interposer layers are known as grids which are numbered as shown in Figure 8.

Every four routers from the processing layer are connected to one specific router from the lower layer, so the number of interposer routers is one-quarter of the processing layer. Figure 9 shows routing mechanism of our proposed method.

We propose that using virtual channels at the interposer layer may be an effective solution to allocate two different paths for each type of traffic. In Figure 10, it is clearly presented.

When a core wants to send a packet from the processing layer to the interposer layer in the light of the load-balancing mechanism, in a grid attached to that core, it is decided which virtual channel should be selected to go through. This may be possible just by considering the address of destination which that packet carries. Since we have 64 cores in the upper layer, if the destination address is more than 63, this is synonymous with the proof that the packet is CtC and should be transmitted throughout VC1 in the interposer layer. This channel segregation in the interposer layer in which two different types of traffic may be differentiated is the solution to avoid the possible conflict as well as potential congestion.

An appropriate channel is initially chosen based on arbitration and switching mechanism [29]. As it can be seen in Figure 9, if we look at a tile in detail, first, according to the destination grid address, it is decided in arbitration block which virtual channel should be selected. Then, in switching part, through input/output ports, the coming packet goes ahead through the ideal virtual channel which is either VC1 associated to CtC or VC2 connected to CtM. This selection happens once the packet is sent from the upper layer to the lower layer.

Figure 11 demonstrates our proposed method with an example. Considering the CtC traffic, node 21 aims to send its packet to node 49. In the case of using the load-balancing mechanism, after detecting congestion in the processing layer, this packet should be sent to grid 6 connected to node 21 in the interposer layer. Once it reaches to grid 6, according to its destination address which is less than 63, this packet is associated to the CtC traffic. This means that it is supposed to go back to the upper layer to node 49, its destination, after crossing grids 5, 4, 8, and 12, respectively, on the light blue path.

At the same time, node 26 wants to send its packet to a memory block located in the west-south corner. Indeed, it should be gone to node 82 which is an interface node connected to the memory block. As it may be seen, its destination address is higher than 63 so it comes from a CtM traffic and uses the interposer layer anyway. The packet belongs to node 26 which goes through grids 5, 4, 8, and 12, respectively, on the dark blue path. Using VCs in the interposer layer provides different types of traffics not to meddle with each other. This will improve the network latency when packets are less when waiting to reach their destination either through the processing core or memory block.

After dedicating a specific virtual channel to the CtM traffic, it is proposed that memory-related packets may be separated as well according to the memory block numbers they want to interact with. We propose multiple links in the interposer layer for the virtual channel allocated to the CtM traffic. For deciding on what link is the best to choose for CtM packets, we number memory blocks as shown in Figure 12.

The reason for this numbering is related to our proposed way shown in Figure 13. This is actually a demultiplexing method. In Table 1, grids connected to memory interface have been presented alongside the number of memory blocks

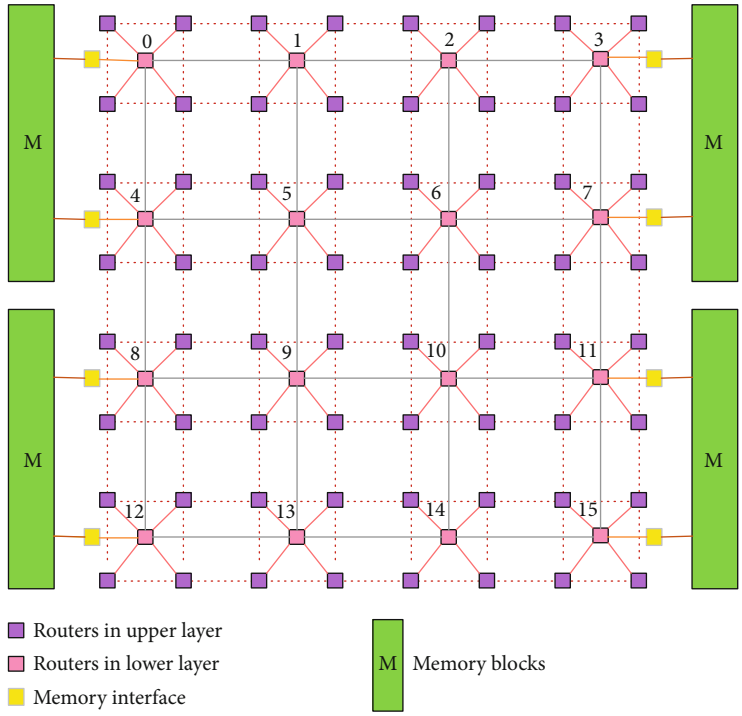


Figure 8: Illustration of grid numbers.

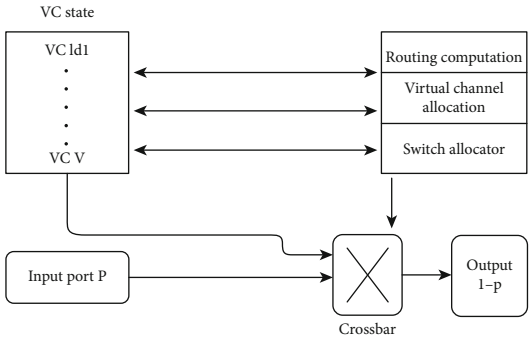


Figure 9: Proposed routing mechanism.

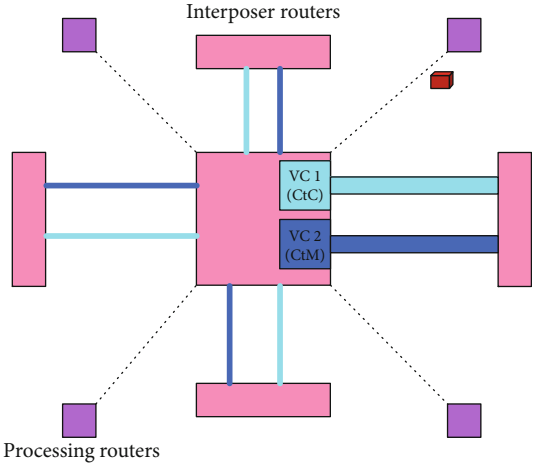


Figure 10: Using specific virtual channels in order to separate different types of traffic.

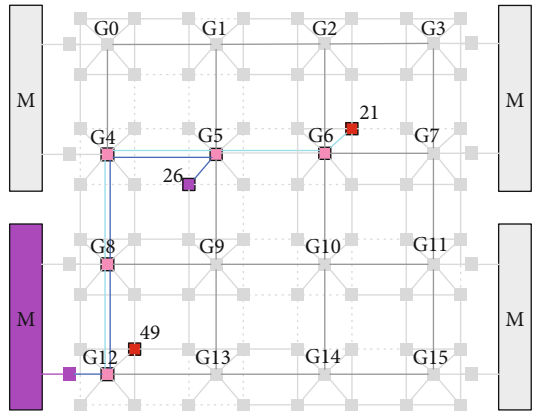


Figure 11: An example of how specific virtual channels may lead to the traffic segregation.

related to each of them both in a decimal format (Table 1(a)) and a binary format (Table 1(b)). In the binary format, there is a relationship between grid numbers and memory block numbers which we numbered accordingly. As it may be seen in Table 1(b), if we put the first and the last bit of grid numbers together, this will build the memory block numbers. That is, when a packet reaches to an interposer router and go along with its specific virtual channel, it may be decided on a specific link to go through to a particular memory block. This parallel transferring enables CtM packets to reach their memory block without any conflict with other packets either CtC or other CtM ones.

This is implemented by a demultiplexer whose selection links are the first and the last bits of destination grid address as shown in Figure 13. For example, consider a CtM-related packet with the destination grid ID of 11, which is supposed

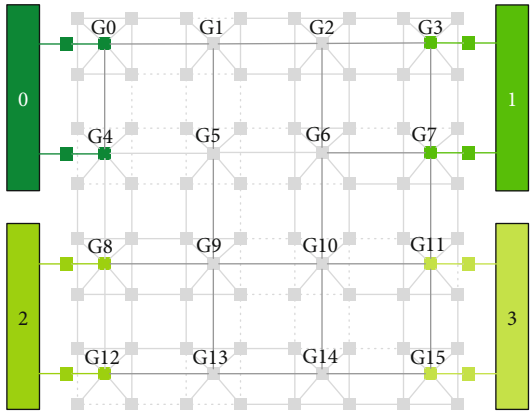


Figure 12: Allocating specific numbers to each group of memory blocks.

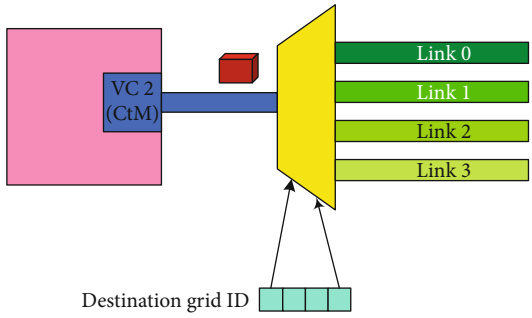


Figure 13: The demultiplexing of memory-related packets.

to transfer through the interposer layer, anyway, reaches to the VC2 (allocated to CtM traffic). In this step, the first bit and the last bit of 11 in the binary format shown in Table 1 determine which link is the best to move. In this example, it is link 3. The function of demultiplexer may be implemented in two different ways: (1) put a physical demultiplexer beside CtM virtual channel or (2) manage the function of division by controlling the time throughout each packet that wants to move to reach its specific memory block. In this paper, in order to diminish overhead, we consider the latter strategy.

We describe our proposed method in a pseudocode form as shown in Figure 14.

3. Experiments

To perform our evaluation, we used a cycle accurate interconnection network simulator, BookSim [31]. In order to simulate our proposed structure, we altered some characteristics of this simulator like workload trace and the file related to the network. A broad range of workloads have been utilized so as to size up our strategy and other methods such as the traditional load-balancing method and a scenario in which there is no load-balancing strategy. Table 2 contains the basic parameters related to the simulation.

3.1. Coherence and Memory Traffic Analysis. In order to evaluate the influence of different types of traffic, we simulated various proportions of memory traffic and processing layer ones.

Table 1: The connection between grids and memory blocks in (a) decimal and (b) binary format. Figure (b) presents the relationship between grids' numbers and related memory block numbers.

(a)

Grid (Connected to Memory Intereface) Numbers	Related Memory Block Numbers
0	0
3	1
4	0
7	1
8	2
11	3
12	2
15	3

(b)

Link	Edge Grid numbers in Bin				Memory block numbers in Bin
Link 0	0	0	0	0	00
	0	1	0	0	00
Link 1	0	0	1	1	01
	0	1	1	1	01
Link 2	1	0	0	0	10
	1	1	0	0	10
Link 3	1	0	1	1	11
	1	1	1	1	11

$Total\_Latency = Packets\ numbers \times Latency\ for\ one\ Packet$
$If\ (Total\_Latency \geq Threshold)$
{
Phase 1: *Packet ejected from upper router => connected grid*
In connected grid:
1) *Arbitration*: *evaluation of the header flit including the final destination*
$If\ (destination\ address \leq 63)\ then\ i = 0$
else $i = 1$;
2) *Switching*: *choose the* VC_i
3) *Routing function*: *XY-based routing*
Phase 2: *case* VC_2 *with*
Link 0 *if Grid ID* = 0,4;
Link 1 *if Grid ID* = 3,7;
Link 2 *if Grid ID* = 8,12;
Link 3 *if Grid ID* = 11,15;

Figure 14: The pseudocode related to our method.

For this, disparate injection rates have been used so as to yield better results. The results are shown in Figure 15;

We evaluated five various distributions among memory and processing cores. As it may be seen in Figure 15, when the percentage of the memory traffic increases, the latency of the network rises accordingly. There is also an increase

Table 2: Simulation parameters.

Primary factors	
Injection rate	0.01, 0.5, and 0.1
Processing die topology	Mesh 8 × 8
Interposer topology	Mesh 4 × 6
Coherence traffic	Uniform, neighbor, randperm, and hotspot
The size of packet	5 flits
Memory traffic percentages	25%, 30%, 40%, and 50%
Percentages of traffics	1: 25% memory traffic, 75% CPU traffic 2: 35% memory traffic, 65% CPU traffic 3: 45% memory traffic, 55% CPU traffic 4: 55% memory traffic, 45% CPU traffic
Virtual channels for the interposer layer	2 VCs

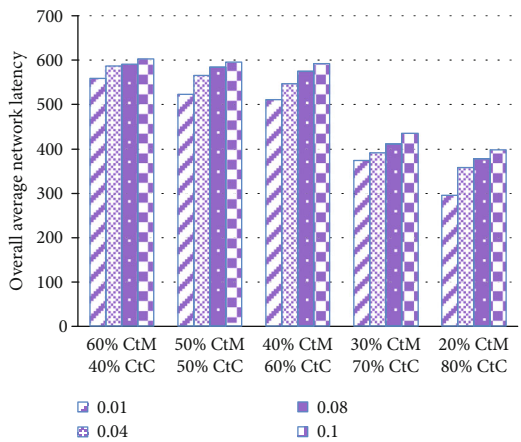


Figure 15: Various percentages of memory traffic and CPU traffic. Each bar represents latencies of different injection rates.

in the amount of latency when the rate of injection increases. The highest overall average latency is associated with the highest CtM traffic (60%), and the highest injection rate which is 0.1.

3.2. Performance Comparison. In this paper, we try to simulate various percentages of both the coherence traffic and the memory traffic. Four different states have been considered as shown in Figure 15: (a) 25% CtM and 75% CtC traffic, (b) 35% CtM and 65% CtC traffic, (c) 45% CtM and 55% CtC traffic, and (d) 55% CtM and 45% CtC.

First, the results bring about that the more proportion of traffic dedicated to memory, the more network latency is expected.

Second, in a scenario that there is no load-balancing mechanism, the overall average latency is the highest in four states. This is rational because coherence traffic is only transmitted on the processing layer and not allowed to use another alternative layer to move (the interposer layer); likewise, the lower layer is merely devoted to memory-related traffic. This is not flexible, and once the coherence traffic exceeds from a certain threshold, this is susceptible to be a bottleneck due to the potential congestion. Finally, as the packets injected to the network tend to increase, either CtM or CtC, the overall average network latency rises.

According to what was explained above and considering Figure 15, our proposed method, which uses virtual channels and multiple links to hinder the interference brought by the load-balancing mechanism, improves the latency compared to the unbalancing and conventional load-balancing mechanisms. Figure 16 demonstrates how the aforementioned scenarios affect the overall network latency with three different cases of memory traffic. We also analyzed our proposed strategy on the other types of traffic such as uniform, randperm, hotspot, and neighbor.

As it can be observed in Figure 17, the hotspot traffic pattern has far more latency in comparison with other types of traffic. The rationale behind this is that when this traffic pattern is distributed upon the processing layer, some nodes will be susceptible to face bottleneck. This, therefore, makes these nodes to be saturated for some instances which affects the overall latency and culminates in the load-balancing strategy as well. Primarily, due to the fact that specific virtual channels provide packets with disparate links as soon as they reach their connected grid at the interposer layer, they are supposed to go through their unique path which is different from the paths of other types of traffic use. This is synonymous with no conflict and no interference although accompanied by a little overhead. Using virtual channels and multiple links at the lower layer, the interposer routers face the physical overhead which may be downplayed by a considerable latency improvement. Supplementary to this, as virtual channels are the internal feature of routers, using them are not even considered an overhead.

3.3. Network Utilization and Efficiency. When using the load-balancing mechanism in 2.5D-based network-on-chip, the underutilized resources, namely, routers at the interposer layer are effectively utilized. Simply put, we have two network layers: the processing layer and the interposer layer. If congestion occurs at the top layer, most of the upper routers involve in the process of data stream are exploited, whereas the interposer routers have little function to do. This, therefore, culminates in network utilization in the light of the load-balancing method. Likewise, our proposed method enhances efficiency and network utilization at the interposer layer throughout using the load-balancing mechanism. Packets sent from the processing layer to the interposer layer are managed and controlled in our method, which enables the network not to be overwhelmed with different types of traffic. We have specified two kinds of virtual channels to avoid data conflict arising from various traffic flows, which are CtC and CtM. As well as this, multiple links at the interposer layer facilitate the way of distributing CtM packets to their particular memory which focuses mainly on the interposer network utilization and its efficiency.

Integrated multicore microprocessor with 2.5D silicon interposer, memory, and accelerator, which offers the advantages of flexibility of system integration and energy efficiency in terms of network utilization.

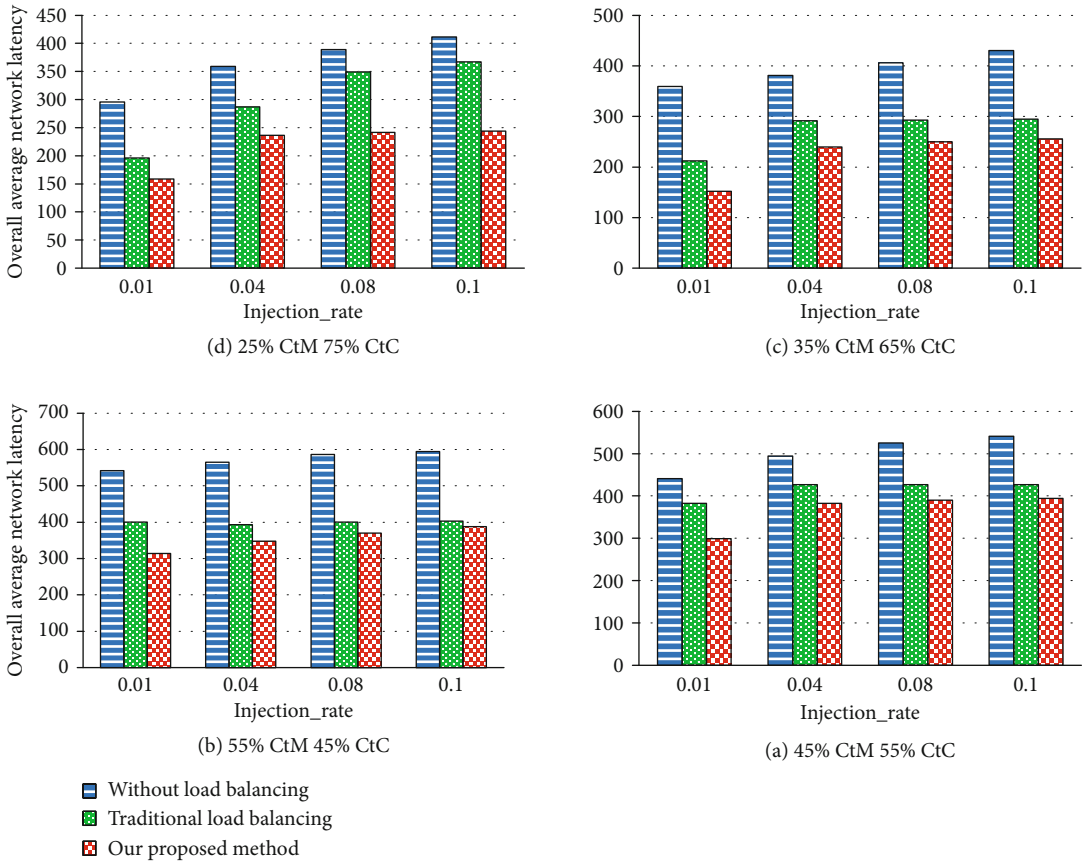


Figure 16: Overall average network latency for three methods: the lack of load balancing, conventional load balancing, and proposed strategy.

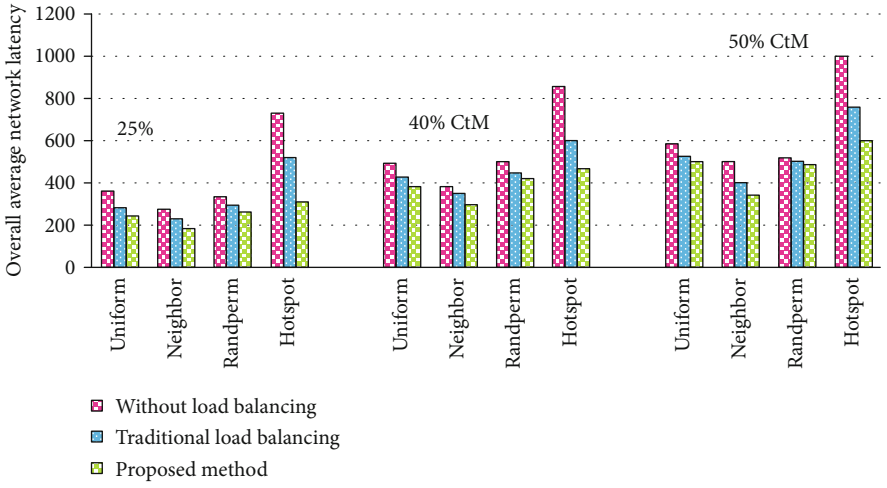


Figure 17: Overall average network latency for three methods: the lack of load balancing, conventional load balancing, and proposed strategy.

4. Conclusions

In order to manage and control interference and conflict caused by the load-balancing strategy in the interposer-based network-on-chip, we proposed a method. When coherence traffic packets are moved from up to down, based on the load-balancing method, a conflict with CtM traffic is expected. The virtual channel-based mechanism proposed in this paper separates the core-to-core traffic from the core-to-memory traffic in the interposer routers and results in no collision between CtM and CtC traffic. The CtM-related virtual channel is also proposed to use multiple links with the aim of classification of memory blocks in parallel. Our proposed strategy has improved the overall average network latency by 32% and 14% in comparison with the case that there are no load-balancing method and conventional

load-balancing mechanism, respectively. This method has been used in smart systems efficiently, and in the future, it will improve real-time applications functionality as well.

Data Availability

Data are available on request through contacting with mona.soleymani@gmail.com.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

References

[1] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "NoC architectures for silicon interposer systems: why pay for more wires when you may get them (from your interposer) for free? in: Microarchitecture (MICRO)," in *2014 47th Annual IEEE/ACM International Symposium on*, pp. 458–470, Cambridge, UK, December 2014.

[2] S. Killge, N. Neumann, D. Plettemeier, and J. W. Bartha, "Optical through-silicon vias," *3D Stacked Chips*, pp. 221–234, 2016.

[3] G. Y. Tang, S. P. Tan, N. Khan et al., "Integrated liquid cooling systems for 3-D stacked TSV modules," *IEEE Transactions on Components and Packaging Technologies*, vol. 33, no. 1, pp. 184–195, 2010.

[4] J. H. Lau, Y. S. Chan, and S. W. R. Lee, "Thermal-enhanced and cost-effective 3D IC integration with TSV (through-silicon via) interposers for high-performance applications," in *Proceedings of the ASME 2010 International Mechanical Engineering Congress and Exposition. Volume 4: Electronics and Photonics*, pp. 137–144, Vancouver, British Columbia, Canada, January 2010.

[5] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert, "Interconnect memory challenges for multi-chip, silicon interposer systems," *Proceedings of the 2015 International Symposium on Memory Systems - MEMSYS '15*, pp. 3–10, 2015.

[6] S. Dadashi, M. Reshadi, A. Reza, and A. Khademzadeh, "An expandable topology with low wiring congestion for silicon interposer-based network-on-chip systems," *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 12, 2019.

[7] S. Dadashi, M. Reshadi, A. Reza, and A. Khademzadeh, "Decreasing latency considering power consumption issue in silicon interposer-based network-on-chip," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7646–7664, 2019.

[8] C. Li, S. Ma, L. Wang, Z. Wang, X. Zhao, and Y. Guo, "DLL: a dynamic latency-aware load-balancing strategy in 2.5D NoC architecture," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 646–653, Scottsdale, AZ, USA, October 2016.

[9] G. Kumar, T. Bandyopadhyay, V. Sukumaran, V. Sundaram, S. K. Lim, and R. Tummala, "Ultra-high I/O density glass/silicon interposers for high bandwidth smart mobile applications," in *2011 IEEE 61st Electronic Components and Technology Conference (ECTC)*, pp. 217–223, Lake Buena Vista, FL, USA, May 2011.

[10] T. G. Lenihan and E. J. Vardaman, "Challenges to consider in organic interposer hvm," *TechSearch International for iNEMI Substrate & Packaging Workshop, Toyama*, 2014.

[11] M. Odeh, B. Voort, A. Anjum, B. Paredes, C. Dimas, and M. S. Dahlem, "Gradient-index optofluidic waveguide in polydimethylsiloxane," *Applied Optics*, vol. 56, no. 4, pp. 1202–1206, 2017.

[12] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in network-on-chip," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pp. 203–214, Salt Lake City, UT, USA, February 2008.

[13] J. Kim, C. Nicopoulos, D. Park et al., "A novel dimensionally-decomposed router for on-chip communication in 3D architectures," *Proceedings of the 34th annual international symposium on Computer architecture - ISCA '07*, 2007.

[14] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and management of 3D chip multiprocessors using network-in-memory," in *33rd International Symposium on Computer Architecture (ISCA'06)*, pp. 130–141, Boston, MA, June 2006.

[15] D. Park, S. Eachempati, R. Das et al., "MIRA: a multi-layered on-chip interconnect router architecture," in *2008 International Symposium on Computer Architecture*, pp. 251–261, Beijing, China, June 2008.

[16] A. Zia, S. Kannan, G. Rose, and H. J. Chao, "Highly-scalable 3D CLOS NOC for many-core CMPs," in *NEWCAS Conference*, pp. 229–232, Montreal, Mayada, June 2010.

[17] Y. Xu, Y. Du, B. Zhoo, X. Zhou, Y. Zhang, and J. Yang, "A low-radix and low-diameter 3D interconnection network design," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 30–42, Raleigh, NC, February 2009.

[18] S. Volos, C. Seiculescu, B. Grot, N. K. Pour, B. Falsafi, and G. De Micheli, "CCNoC: specializing on-chip interconnects for energy efficiency in cache coherent servers," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pp. 67–74, Lyngby, Denmark, May 2012.

[19] J. Xie, J. Zhao, X. Dong, and Y. Xie, "Architectural benefits and design challenges for three-dimensional integrated circuits," in *2010 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 540–543, Kuala Lumpur, Malaysia, December 2010.

[20] C. Li, S. Ma, L. Wang, Z. Wang, X. Zhao, and Y. Guo, "Overcoming and analyzing the bottleneck of interposer network in 2.5D NoC architecture," in *Advanced computer Archi- tecture: 11th Conference, ACA 2016*, pp. 40–47, Weihai, China, August 2016.

[21] M. Jackson, "A silicon interposer-based 2.5D-IC design flow, going 3D by evolution rather than by revolution," in *2011 IEEE International 3D Systems Integration Conference (3DIC), 2011 IEEE International*, Osaka, Japan, January 2012.

[22] J.-H. Huang, *NVidia GPU technology conference: keynote*, 2013.

[23] K. Saban, *Xilinx stacked silicon interconnect technology delivers breakthrough FPGA capacity, bandwidth, and power efficiency*, Xilinx, White Paper, United States, 2011.

[24] M. J. Santarini, "Stacked and loaded: Xilinx SSI, 28-Gbps I/O yield amazing FPGAs," *Xcell Journal*, vol. 74, pp. 8–13, 2011.

[25] H. Shabani and X. Guo, "Clus Cross: a new topology for silicon interposer-based network-on-chip," *Proceedings of*

13th IEEE/ACM international symposium on network-on-chip, 2019.

[26] D. Stow, I. Akgun, and Y. Xie, "Investigation of cost-optimal network-on-chip for passive and active interposer systems," in *2019 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, Las Vegas, NV, USA, USA, June 2019.

[27] Y. Li and L. Chen, "Equi Nox: Equivalent Noc injection routers for silicon interposer based throughput processors," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, San Diego, CA, USA, USA, February 2020.

[28] M. Soleymani, M. Reshadi, N. Bagherzadeh, and A. Khademzadeh, "CLBM: controlled load-balancing mechanism for congestion management in silicon interposer NoC architecture," *Journal of System Architecture*, vol. 98, pp. 102–113, 2019.

[29] C. Liu, L. Zhang, Y. Han, and X. Li, "Vertical interconnects squeezing in symmetric 3d mesh network-on-chip," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pp. 357–362, Yokohama, Japan, January 2011.

[30] R. Das, S. Narayanasamy, S. K. Satpathy, and R. Dreslinski, "Catnap," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 320–331, 2013.

[31] N. Jiang, D. U. Becker, G. Michelogiannakis et al., "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 86–96, Austin, TX, USA, April 2013.