DLL: A Dynamic Latency-Aware Load-Balancing Strategy in 2.5D NoC Architecture

Chen Li, Sheng Ma*, Lu Wang, Zicong Wang, Xia Zhao[1], Yang Guo
College of Computer, National University of Defense Technology, Changsha, China
[1]ELIS department, Ghent University, Belgium
Email: {lichen,masheng,luwang,wangzicong,guoyang}@nudt.edu.cn [1]xia.zhao@Ugent.be *Corresponding Author

***Abstract*—As the 3D stacking technology still faces several challenges, the 2.5D stacking technology gains better application prospects nowadays. With the silicon interposer, the 2.5D stacking can improve the bandwidth and capacity of the memory system. To satisfy the communication requirements of the integrated memory system, the free routing resources in the interposer should be explored to implement an additional network. Yet, the performance is strongly limited by the unbalanced loads between the CPU-layer network and the interposer-layer network.**

In this paper, to address this issue, we propose a dynamic latency-aware load-balancing (DLL) strategy. Our key innovations are detecting congestion of the network layer via the average latency of recent packets and making the network layer selection at each source node. We leverage the free routing resources in the interposer to implement a latency propagation ring. With the ring, the latency information tracked at destination nodes is propagated back to source nodes. We achieve load-balance by using these information. Experimental results show that compared with the baseline design, a destination-detection strategy and a buffer-aware strategy, our DLL strategy achieves 45%, 14.9% and 6.5% of average throughput improvements with minor overheads.

I. Introduction

Recently, the silicon interposer-based stacking, known as “2.5D stacking” [1], is gaining more traction [2]. As shown in Figure 1, the 2.5D stacking technology stacks multiple dies side-by-side on a silicon interposer carrier. While the 3D-stacking approach is a revolutionary approach which needs new co-design and methods for design flow and testing, the 2.5D-stacking approach is evolutionary [3]. It side-steps many challenges in the 3D stacking technology and is well supported by current design tools. In addition, the current stacked memory system provide a fixed bandwidth and capacity per memory stack. Since the silicon interposer has enough area to integrate more memory stacks than 3D stacking, it achieves larger memory capacities and higher memory bandwidth [2]. Therefore, commercial 2.5D stacking products have become popular during the past few years [4] [5] [6].

In order to efficiently satisfy the communication requirement of the 2.5D stacking system, previous work [7] [2] [8] shows that abundant free routing resources inside the silicon interposer should be exploited to implement an additional network. When there are two network layers on a 2.5D silicon interposer system, it is desirable to differentiate the core-to-core coherence traffic and the core-to-memory traffic [7]. As these two types of traffic have different characteristics,

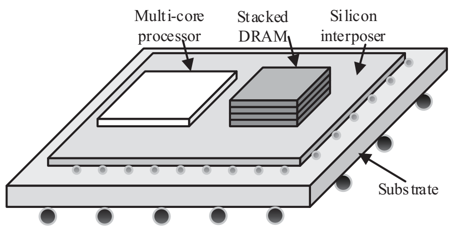


Fig. 1. The 2.5D stacking technology

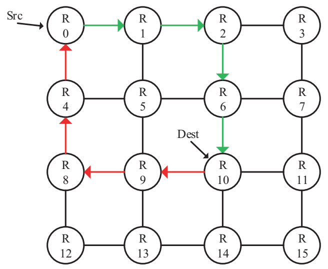


Fig. 2. Destination detection load-balancing strategy (Assuming XY dimensional order routing is used, the green path is the congestion collected path; the red path is the packet sending path of node 10. The congestion status of the green path is regarded as the congestion status of the red path for packets sending at node 10.)

numerous benefits will be received to transfer these different types of traffic on different network layers.

However, we run PARSEC benchmarks on a baseline 2.5D architecture and find that the memory traffic accounts for average 14.8% of the total traffic. These workloads have a strong bias toward core-to-core traffic over core-to-memory traffic. The network layer for core-to-core traffic is congested, while the other layer is underutilized. It results in poor performance. **In consequence, a load-balancing strategy is demanded in the 2.5D Network-on-Chip (NoC) architecture to balance the workload on the two network layers and maximize the resources utilization.**

Buffer-aware and latency-aware methods are common methods to achieve load-balance in multi-layer networks. The traditional buffer-aware method collects the congestion information based on the buffer occupancy of each neighbor router [9] [10]. However, as this local congestion information cannot fully reflect the global congestion status, it is not suitable for the network selection on the 2.5D NoC. As we know, the destination detection latency-aware method is the only load-balancing strategy on the current 2.5D NoC [7]. It tracks the latency of received packets to detect congestion and select the network layer at destination nodes. However, this strategy uses inaccurate information, because packet receiving paths are different with packet sending paths for each node; it uses the latency of receiving paths to infer the latency of sending paths. For example, in Figure 2, node 10 collects the latencies of received packets from the green path as the congestion information. Yet, its packets are sent along the red path.

We need to study appropriate congestion detection mechanisms to obtain accurate network status and make efficient network selection. It is fundamental to achieve load-balance for the 2.5D NoC architecture.

In this paper, we propose a dynamic latency-aware load-balancing (DLL) strategy for the 2.5D NoC architecture. The DLL strategy first collects congestion information at each destination node. Then, a latency propagation ring is designed to send these congestion information back to source nodes. Finally, each source node selects a network layer for packet transferring according to the congestion information. Our DLL strategy makes the following primary contributions:

- We evaluate and analyze the traffic of PARSEC benchmarks, and find that the communication traffic on the 2.5D NoC has a strong bias toward one network layer over the other one. It boosts the demand for an efficient load-balancing strategy.
- We analyze the drawbacks of current load-balancing strategies. The buffer-aware method cannot indicate the global network congestion status. The current latency-aware method cannot indicate the congestion status accurately.
- We propose a dynamic latency-aware load-balancing strategy in the 2.5D NoC. This strategy collects congestion information at destination nodes, and send back these information to source nodes for the network selection. A multi-link and bufferless ring is designed to propagate the congestion information.
- Compared with a baseline 2.5D NoC without the load-balancing strategy, a current latency-aware method and a buffer-aware method, our strategy improves the average throughput by 45%, 14.9% and 6.5%, respectively, while the overhead is very low.

The rest of the paper is organized as follows. We introduce related work in Section II. Section III presents the design of our proposed strategy, including the target system, design principles and design details. In Section IV, we report experimental results. In Section V, we give further discussion and finally we conclude in Section VI.

II. Related work

As the 2.5D stacking technology exhibits notable advantages in the bandwidth, latency and cost, a series of 2.5D stacking products have been launched in industry recently. The first 2.5D stacking commercial product is Xilinx Virtex-7 2000T FPGA [5], which stacks four smaller FPGAs into a silicon interposer. High Bandwidth Memory (HBM) [11] and Hybrid Memory Cube (HMC) [12] are two emerging stacked memory technologies and they are widely leveraged by 2.5D stacking processors. Currently, they can provide 128GB/s available bandwidth per stack. AMD Radeon R9 Fury X^{new} GPU features HBM with the 2.5D stacking technology to provide large bandwidth and capacities [4]. The NVidia Pascal GPU architecture features the second-generation HBM as well [6] [13]. In addition, the second-generation HMC will be adopted with 2.5D stacking in the next-generation Intel Xeon Phi Co-Processor [14].

Enright Jerger et al.'s work is the first one to explore the design space of interposer-based NoC organizations [7]. They compare several topologies and propose a load-balancing strategy. However, their strategy tracks congestion information (latency) at destination nodes, and adopts these information for network selection at destination nodes as well. Since the packet sending path is different from the packet receiving path, this strategy uses inaccurate network status to make network selections. We refer this strategy as DestDetect in this paper.

There are numerous proposals for congestion control in the traditional 2D NoC. Li et al. propose DyXY routing based on local network congestion status [15]. It achieves higher performance compared with the static routing. RCA [10] is the first work to utilize both local and non-local information to improve load balance. However, it introduces interference during the congestion calculation. DBAR [16] addresses this issue by offering the dynamic isolation when collecting the congestion information. All these methods adopt statistics of buffer usage as the congestion information. It benefits performance on the routing path selection rather than the network selection.

In the aspect of load-balance in the 3D NoC and the multi-NoC, Ranmanujam et al. present an efficient Layer-Multiplexed (LM) 3D architecture [17]. They replace one-layer-per-hop routing in a conventional 3D network with vertical de-multiplexing and multiplexing structures. The network layer selection is achieved by using a set of flit-counters for each ordered pair of input and output ports in the load-balancing logic. The congestion information reflects the communication traffic load on the network layer, but the hotspot area in the network layer cannot be found. Catnap [9] is an energy-efficient multi-network and its subnetworks can be power gated without compromising the network connectivity. They implement a subnet-selection policy based on the observation of the congestion status. Buffer occupancy of input buffer in each router is calculated to detect congestion. Nevertheless, this method cannot fully reflect the global congestion status. We refer this congestion detection and network selection strategy as LocalBuf strategy, and compare it with our proposed DLL strategy.

III. DLL Strategy

In this section, we first introduce our 2.5D NoC target system. Then we analyze the communication and congestion features of the NoC to explain our design motivation. Last, we present the DLL strategy in two aspects. One is the congestion detection and network selection scheme and the other one is the latency propagation ring.

A. Target System

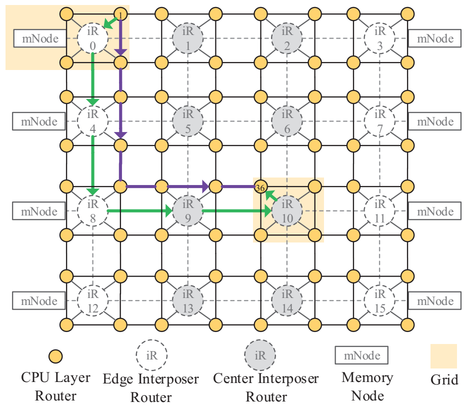
As the DLL strategy is broadly applicable to a wide scope of interposer-based systems and multi-network selection, we illustrate an example in this paper to make our strategy more concrete. Our 2.5D interposer-based system integrates a 64-core CPU and 4 stacked DRAMs around on a silicon interposer [7]. There are two network layers in our system, the upper layer is the CPU layer and the lower layer is the interposer layer. These two network layers are connected by TSVs and μbumps. As shown in Figure 3, the topology of our 2.5D NoC architecture is the mesh on the CPU layer. Since the μbump induces significant overheads [18], we use the concentrated mesh on the interposer layer to reduce the router count. Each of the 16 interposer nodes connects four CPU nodes. There are a total of 8 memory controllers on left and right sides of the interposer network. Each one has two DRAM channels and connects a nearby interposer node. Figure 3 also shows two types of interposer implementations, including the active type and the passive type. Compared with the active interposer, the passive interposer has no active device (logics/gates), and all interposer routers are implemented in the CPU layer. In the near term, the passive interposer is a practical way, while the active interposer is more likely to be a 3D integrated way.

There are two types of traffic, including the core-to-core traffic and the core-to-memory traffic. Previous study shows that differentiating these two types of traffic on the two network layers has several advantages [7]; the core-to-core coherence traffic is transferred on the CPU layer and the core-to-memory traffic is transferred on the interposer layer. However, this partitioning of NoC traffic is not strict. It depends on both application needs and the actual NoC usage. In following sections, we will introduce our load-balancing strategy. It partitions the NoC traffic based on the congestion status.

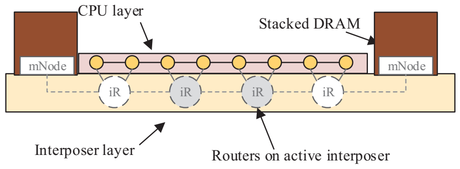
B. Design Principles

We analyze the communication and congestion features of NoC. These features are quite important for designing a load-balancing strategy. According to the communication and congestion features, we also conclude two design principles for the load-balancing strategy in the 2.5D NoC architecture.

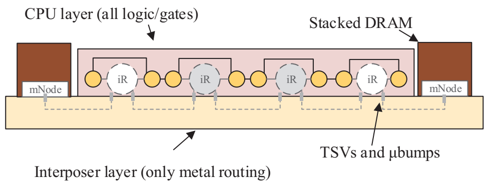
- **Communication Feature:** Many applications, including the PARSEC and SPLASH-2 benchmarks, exhibit less uniform, but heavy communication traffic between several pairs of nodes in the spatial behavior [19]; most communication packets are transferred between several pairs of nodes, similar to synthetic permutation traffic patterns. Communication paths with heavy-load traffic can be reflected clearly by latencies of packets.
- **Design Principle 1:** The congestion metric of buffer occupancy rate in each router reflects the congestion status of the router. Only the local congestion status can be detected by each node. On the contrary, average latencies of recent packets reflect the path status in the network. Thus, global congestion status on a network layer can be detected more accurately by the latency-aware method. We adopt the latency-aware method for congestion detection rather than the buffer-aware method.
- **Congestion Feature:** The NoC communication traffic exhibits permutation-like traffic patterns. Routing paths with heavy workloads can easily become congested, while routing paths with light workloads are under utilization. Nodes through heavy-load routing paths regard the whole layer of network as congested, while nodes through light-load routing paths regard the whole layer of network as under congested. Thus, observed from different nodes, the congestion status of a network layer is different.
- **Design Principle 2:** As different congestion status of a network layer are observed from different nodes, we should use the congestion information observed from source nodes for network selection. If it is used for network selection at destination nodes, inaccurate congested paths are reflected. If it is used with a global view, the non-uniform congestion status of the whole network layer cannot be reflected. Light loaded paths in the network



(a) Top view



(b) Side view (active interposer)



CPU layer (all logic/gates)
Stacked DRAM
mNode
iR
TSVs and μbumps
Interposer layer (only metal routing)

(c) Side view (passive interposer)

Fig. 3. The 2.5D NoC target system

layer may be regarded as congested paths or vice versa. Therefore, network selection is better made at source nodes, rather than destination nodes or a global view.

In this paper, we leverage latency-aware method to detect global layer congestion and select the network layer at each source node in the 2.5D NoC architecture.

C. Congestion Detection and Network Selection

Under the guidance of the design principles, we develop our DLL strategy which adopts latencies of recent packets as the metric of the network congestion, and compares average packet latencies of two network layers for network selection at source nodes. For example, in Figure 3(a), if node 1 is sending a packet to node 36, node 1 first compares the average latencies of sending packets for the two network layers. If the average latency on upper network layer exceeds the lower network layer by a certain threshold, the packet will be routed through the green path, rather than the purple path, for load-balance.

Technically, our DLL strategy needs to solve three essential problems: 1) How to record latencies of packets at routers? 2) How to generate and propagate congestion information? 3) How to make an effective network selection according to the congestion information?

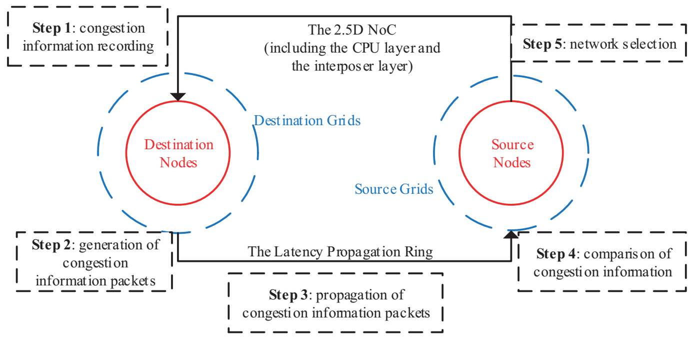
There are four detailed design aspects in the DLL strategy.

First, since packets transferring with far distances take longer time than packets with short distances, the total latencies between packets is unfair and mislead us to make a poor network layer selection. We compare the latency per hop between packets to select the network layer. Latency information are recorded in the head flit of each packet.

Second, if we propagate latencies of all packets back to the source nodes, the overhead will be intolerable. Thus, we adopt a coarse-grained method. The network is partitioned into several grids. Each grid composed of multiple nodes, and the congestion information of nodes in the same grid will be merged to reduce the overhead. As shown in Figure 3(a), if node 1 sends a packet to node 36, our coarse-grained strategy considers the packet is sent from grid 0 to grid 10. We assign 4 CPU nodes connected to an interposer node into a grid, as the yellow block shown in Figure 3(a). Memory nodes are also included in edge grids. The grid number is the same with the interposer node number.

Third, as all memory controllers are located on two horizontal sides of the interposer network, edge sides of the lower network may become bottleneck in the 2.5D NoC. The YX-Z routing instead of the XY-Z routing is leveraged to reduce traffic pressure of edge sides. In this way, packets go through as less edge routers as possible to destination memory controllers.

Fourth, the DLL strategy eliminates both protocol-level and network-level deadlock. Protocol-level deadlock is avoided by leveraging virtual channels. Dimension-ordered routing is leveraged to avoid network-level deadlock. Even if packets change its transferred network layer, deadlock-freedom is still maintained, as the Z hop is the first hop or the last hop of packets. No cycle can be formed.



Step 1: congestion information recording
The 2.5D NoC (including the CPU layer and the interposer layer)
Step 5: network selection
Destination Nodes
Destination Grids
Source Nodes
Source Grids
Step 2: generation of congestion information packets
The Latency Propagation Ring
Step 4: comparison of congestion information
Step 3: propagation of congestion information packets

Fig. 4. The procedure of the DLL strategy

Source grid	Average hop latency	Transferred layer
4 bits	4 bits	1 bits

Fig. 5. Congestion information packet

There are 5 steps for congestion detection and network selection, as shown in Figure 4.

- The first step is recording latencies of packets at destination nodes. We add a field in each packet to record the latency. Latencies are tracked by the clock in each router. Once the head flit is ready to be sent out, the field is updated as follows:

$$L = \frac{L_{last} \times hop + t_{out} - t_{in}}{hop + 1}, \tag{1}$$

  where L_{last} is the average hop latency until the last hop, L is the average hop latency until the current hop, hop is the current hop count, t_{out} is the clock of the local router when sending out, t_{in} is the clock of the local router when receiving the header flit. When receiving a packet, the destination node first calculates **average latency of each hop** for the packet. *If the per hop latency is larger than 15 (2'b1111), it will be recorded as 15 (2'b1111).* We believe that 15 cycles per hop is enough to indicate congested status. As the per hop latency has only 4 bits, the calculation is simple and fast.
- The second step is generating the congestion information packet at destination grids. Three types of data are encapsulated into a congestion information packet with 9 bits as shown in Figure 5, including the source grid, the average hop latency and the transferred layer. The transferred layer is encoded as 0 for the upper layer and 1 for the lower layer. The destination node sends the congestion information packet to the interposer node in the same grid. If buffers in the interposer node are full, newly generated congestion information packets will be dropped. As a coarse-grain method is leveraged, it does not strongly impact on the accuracy of congestion collection.
- The third step is propagating congestion information packets back to source grids. An independent ring is proposed for propagating these packets and it will be detailedly introduced in section III-D.

- The fourth step is calculating and comparing the average latencies of each network layer at source grids. According to the transferred layer, the source grid stores congestion information packets into 2 specific FIFOs. The FIFO has 5 slots; it uses the recent 5 packets' latencies to calculate the average latency of each network layer. According to our experiment, the recent 5 packets' latencies are enough to reflect the current congestion status. Once a FIFO receives a new congestion information packet, the average latency for the network layer is updated. As the network selection is made at each source node, the latencies comparison is implemented in each source grid.
- The last step is selecting the appropriate network layer for transferring at source nodes. We compare the average latencies of two network layers calculated at the previous step, and the comparison result is sent to all nodes in that grid. The DLL strategy dynamically sends the packet into the network layer with lighter load. If the average latency of the CPU layer exceeds the average latency of the interposer layer by a certain threshold and the interposer layer is not congested (the average latency of the interposer layer is less than a certain threshold), the packet sent to the CPU layer will be transferred to the interposer layer until it reaches its destination grid. After reaching its destination grid, the packet will be sent to its destination node on the CPU layer from the interposer node. In other cases, each source node will send packets to its default network layer.

D. Latency Propagation Ring

In order to make network selection at source nodes, congestion information packets should be propagated from destination grids to source grids. As shown in Figure 6, we design a latency propagation ring connecting all grids for propagating congestion information packets. The ring is independent from the existing network so that no interference exists and high efficiency can be gained. The independent ring must meet with two objectives. The first one is high bandwidth. A high bandwidth network allows more congestion information packets to be sent back to source grids to gain more accurate congestion status. The second one is low cost. Since the ring induces additional overheads, the cost must be low enough. To meet with these two objectives, the latency propagation ring is designed to be multi-link and bufferless. As the ring is implemented in the interposer layer whose most area and routing resources are unutilized, its overheads have light impacts on the whole architecture.

In order to get high bandwidth, the latency propagation ring is composed of four 7-bit links. The transferring link is decided by the highest two bits of the source grid ID (the destination grid of the congestion information packet). Figure 7 shows an example switch of Grid 0. If a congestion information packet is injecting at Grid 0, the routing unit first select Link 0 based on the highest 2 bits (2'b00) of the source grid ID. And then these highest 2 bits are dropped, and only the remaining 7 bits of the congestion information packet are transferred on the ring link. As illustrated in Figure 6 and Figure 7, the congestion information packets sent to grids with different colors are transferred on the links with different colors. With 4 links, more congestion information packets can be propagated in parallel.

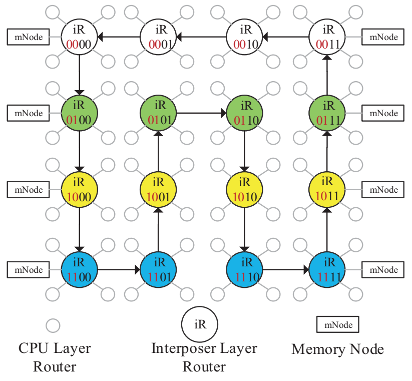


Fig. 6. Latency propagation ring

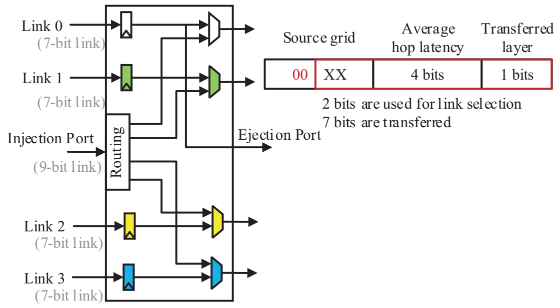


Fig. 7. Latency propagation ring switch of Grid 0

In order to reduce the hardware cost, the latency propagation ring leverages bufferless structure. There is no buffer in each switch. We insert a register on the link path of each switch so that only 2 cycles is taken to transfer a congestion information packet per hop. Once the congestion information packet is injected into the ring, it will reach the source grid without blocking. To eliminate the network-level deadlock, the priority of packet on the ring is higher than the packet waiting for injection. In other words, congestion information packets cannot be injected if there is a packet from the upstream switch.

Transferring congestion information packets induce some delays. Since our proposed latency propagation ring is non-blocking, the delay is not long. The congestion information can timely reflect the network status.

IV. Experiments

We use a cycle accurate interconnection network simulator (Booksim) [20] for the evaluation. We modify Booksim to implement our 2.5D NoC architecture and evaluate different load-balancing strategies with several synthetic traffic patterns. To further verify the result with real applications, we run Netrace with Booksim to evaluate the performance with PARSEC [21]. We use DSENT [22] for area and power evaluations. To efficiently evaluate our proposed DLL strategy, we first analyze bottlenecks of our 2.5D NoC architecture, so that we can understand the performance potential of load-balancing strategies. We also show the benefit of the YX-Z routing compared with the XY-Z routing. Second, we compare the performance of our DLL strategy with the baseline design without a load-balancing strategy and the other two load-balancing strategies. Finally, we discuss hardware overheads of the DLL strategy.

A. Performance Bottleneck Analysis

As the 2.5D NoC architecture leverages the mesh on the CPU layer, the concentrated mesh on the interposer layer and memory nodes are located in two sides, the topology of the whole 2.5D NoC is asymmetric. There are three possible performance bottlenecks of the 2.5D NoC architecture, including the upper layer network, the center portion of the lower layer and the edge portion of the lower layer, as shown in Figure 3. Any of these portions may lead the 2.5D NoC to be saturated, while other network parts are still working in unsaturated state.

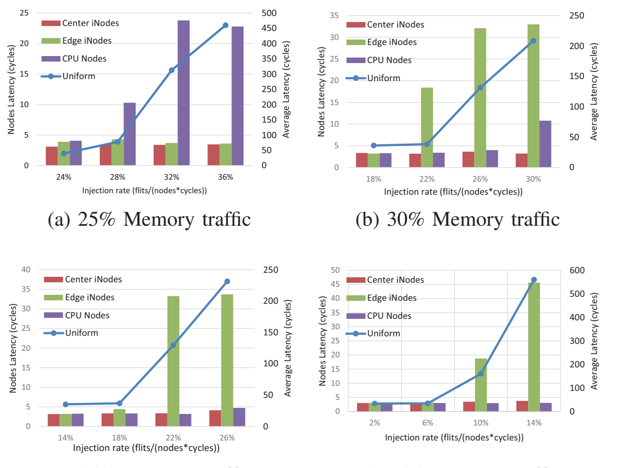
We evaluate average latencies of packets passing through network nodes in these 3 parts. We leverage the XY-Z routing in the baseline design, and results are shown in Figure 8. We find that CPU nodes on the upper layer leads to the saturation of the whole network when the memory traffic accounts for 25% of the total traffic. The bisection bandwidth of the upper layer is two times of the bisection bandwidth of the lower layer. Thus, when the memory traffic occupancy rate is more than 30%, the lower layer becomes the bottleneck. Figure 8 shows that edge nodes on the lower layer leads to the whole network saturation when the percentage of memory traffic is larger than 30%. However, when edge interposer nodes are saturated, latencies of packets passing through center interposer nodes are still low. Even when the memory traffic account for 75% of the total traffic, the edge portion of the lower layer is still the performance bottleneck.

Fortunately, this bottleneck can be alleviated by the YX-Z routing. The YX-Z routing first sends packets along the Y dimension; this can reduce the congestion for edge column routers. Figure 9 shows that when memory traffic is heavy, the YX-Z routing achieves a 56.5% throughput improvement compared with the XY-Z routing. Apparently, since packets are routed in the Y dimension first, the traffic pressure on two edge sides of the interposer NoC is alleviated.

From the above analysis, the load-balancing strategy should be leveraged when the percentage of memory traffic is lower than 30%. We evaluate the PARSEC benchmark and find that memory traffic account for 14.8% of the total traffic. Thus, our strategy can be well used with real applications.

B. Performance Comparison

In this subsection, we compare the DLL strategy with the baseline design (DOR) and other two load-balancing strategies, LocalBuf and DestDetect. The DOR design transfers different types of packets to different network layers. The LocalBuf strategy detects local congestion by the buffer occupancy rate of a router. If the buffer occupancy rate is



(a) 25% Memory traffic (b) 30% Memory traffic

(c) 35% Memory traffic (d) 75% Memory traffic

Fig. 8. Performance bottlenecks (Blue lines describe the average latency on the whole network of the uniform random traffic pattern at different injection rates; the 3 bars describe latencies of packets passing through different nodes at different injection rates.)

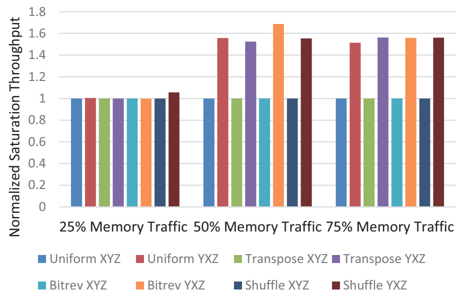


Fig. 9. The XY-Z routing and the YX-Z routing comparison (4 types of traffic patterns on the CPU layer and uniform random pattern on the interposer layer)

larger than 60%, the router is regarded as congested. For the LocalBuf strategy, if more than 1 node of a grid is congested on the upper layer, and the interposer node of the grid is not congested, the coherence traffic is sent to the lower layer for load-balance. The DestDetect strategy is similar to our DLL strategy to leverage the latency as the metric of congestion. It collects latencies of recent packets and calculates average latency per hop at each destination node. Then the destination node directly uses the average latency of received packets to selection the network layer. If the average latency of packets received from the upper layer is larger than the lower layer by a certain threshold, the coherence traffic is transferred to the lower layer. Both the DestDetect and our DLL strategies select the threshold as 8 cycles empirically. All strategies are leveraged in the target system as shown in section III-A. The baseline router has 3 stages pipeline. For fair comparison, the YX-Z routing is used by all strategies.

Figure 10 compares the throughput of these strategies with different percentages of memory traffic. Results show that the average throughput gains of our DLL strategy over the DOR design, the DestDetect strategy and the LocalBuf strategy are

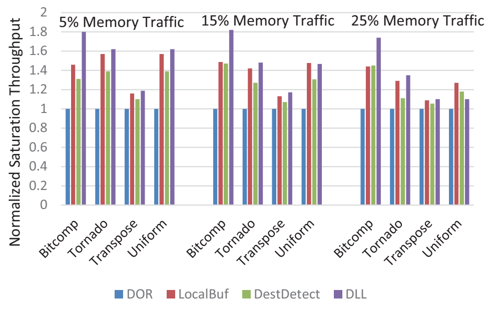


Fig. 10. Throughput

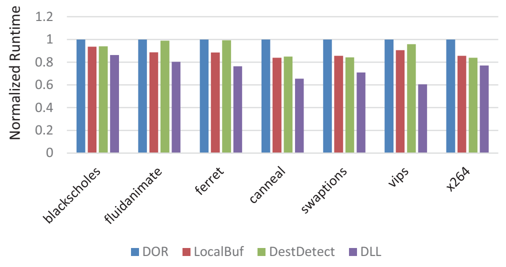


Fig. 11. Performance for PARSEC benchmark

45%, 14.9% and 6.5%, respectively. Focusing on different traffic patterns, our DLL strategy has very limited benefits on the uniform random traffic pattern compared with the LocalBuf strategy. Our DLL strategy performs much better than all other strategies on permutation traffic patterns. It is due to all communications of permutation patterns are between pairs of nodes; they are sensitive to the latency of communication paths. However, the DestDetect strategy performs poorer than the LocalBuf strategy. Latency congestion information collected by the DestDetect strategy are paths for packet receiving, but not for packet sending. Thus the DestDetect strategy is an inaccurate method. Though the LocalBuf cannot reflect congested paths, the local congested information is accurate. It performs better than the DestDetect strategy.

Focus on different percentages of memory traffic. As less percentage of memory traffic leads to larger traffic bias between two network layers, load-balancing strategies get more performance gains with less memory traffic. When the percentage of memory traffic is 5%, the performance of DLL is averagely 55% higher than the performance of DOR design.

We also combine Booksim with Netrace for the real application evaluation. Traces in Netrace are collected from the M5 simulator modeling a 64-core system, which is comprised of in-order cores with a 64KB private L1 cache, a shared, 16MB banked L2 cache, and 8 on-die memory controllers. We evaluate all traces of PARSEC benchmark and find that memory traffic account for 14.8% of the total traffic. It means that load-balancing strategies can be fully used. Results in Figure 11 show that the DLL strategy averagely reduces 26.1% runtime compared with the DOR design, while the LocalBuf strategy performs a little better than the DestDetect strategy.

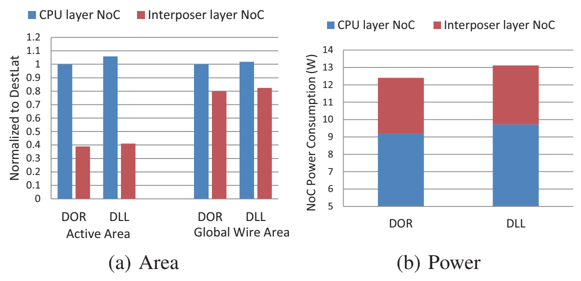


Fig. 12. Hardware overheads

C. Hardware Overheads

Figure 12 shows hardware overheads of the DLL strategy compared with the DOR design. As the LocalBuf strategy and DestDetect strategy induce only a little hardware overhead on logics and buffers, we mainly evaluate the additional overhead of the latency propagation ring. Results are collected using a 45nm bulk/SOI low-V$_t$ process node with an injection rate of 10% flits/node/cycle. There are 8 virtual channels (VCs) and 8 buffers per VC with 128 bits per buffer in each router. The CPU layer NoC and interposer NoC from Figure 12 are logical NoC architectures. Interposer implementations will likely be "passive" in the near-term and no transistors can be provided on the interposer layer. The logical interposer NoC uses the metal interconnects from the interposer layer, while active components (i.e. routers, buffers, repeaters) still reside on the processor die [2]. The deployment of these resources largely increases the performance. Compared with the DOR design, our DLL strategy costs about 7.7% more area and 5.8% more power. In fact, the main additional hardware overheads come from physical links of the latency propagation ring and its associated buffers for recording congestion information packets. These overheads of buffers and physical links are reflected on active area and global wire area, respectively. As our latency propagation ring is designed with bufferless switches, the additional overheads are so small that can be acceptable.

V. Discussions

In this section, we further discuss three issues to give some details in this paper, including topologies, interposer types and the threshold.

First, the 2.5D NoC can apply other topologies. Enright Jerger and Kannan have investigated several topolgies [7] [8]. Double-Butterflies can be used to solve the performance bottleneck of edge portion. This topology reduces the conflict and improves the path diversity of the edge portion for the interposer layer network. In this paper, we use the concentration mesh topology for the interposer layer network. The concentration mesh significantly reduces the overheads of μbumps, and can apply simple routing algorithms to avoid deadlock. This topology acts as an example topology for our design of the load-balancing strategy. The proposed DLL strategy can be used in other topologies, including the double-butterfly.

Second, the interposer is used to stack chips and memories. Current 2.5D chips [4] [23] use the passive interposer without any active devices. It can be implemented in a cheap process technology with high yields. Only metal resources are exploited and the transferring latency can be much lower as metal wires are wide and thick in the interposer. In the far term, the active interposer can be implemented to better exploit resources. In this paper, our goal is to make full use of the metal routing resources in the interposer. As active devices need to be implemented in the chip die, the coarse-grained method and the bufferless ring are leveraged to reduce the overheads of buffers and logics.

Third, setting a suitable threshold is a key point in this paper. The threshold is the difference of average latencies per hop on the two network layers. If the threshold is too large, the network will react slowly to the congestion. If the threshold is too small, wrong judgements of network selection may be made. However, different thresholds are needed in different traffic patterns. In this paper, we choose a relatively suitable threshold through a series of experiments. In the future, we will investigate a feedback scheme to adjust the threshold dynamically.

VI. Conclusions

The 2.5D stacking technology leverages an interposer to stack chips and DRAMs. Making use of the metal resources on the interposer provides fascinating opportunities to explore new features on the 2.5D NoC architecture. In this paper, focusing on the traffic bias between the CPU layer and the interposer layer, we propose a dynamic latency-aware load-balancing (DLL) strategy. It collects average latencies of packets at each destination node and utilizes a multi-link and bufferless ring to propagate latency information back to source nodes for load-balance. We evaluate different percentages of memory traffic to find out network bottlenecks. Experimental results show that our DLL strategy achieves 45%, 14.9% and 6.5% average performance improvements compared with the baseline design, the DestDetect strategy and the LocalBuf strategy, respectively. Meanwhile, it costs about 7.7% more area and 5.8% more power than the baseline design. Future work will extend the DLL strategy with adaptive threshold setting and apply the DLL strategy to other topologies.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (No.61303065, 61672526, 61133007), Doctoral Fund of Ministry of Education (20134307120028).

References

[1] Y. Deng and W. P. Maly, "Interconnect characteristics of 2.5-d system integration scheme," in *Proceedings of the 2001 international symposium on Physical design*. ACM, 2001, pp. 171–175.

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

[3] I. Bolsens and C. Xilinx, "2.5 d ics: Just a stepping stone or a long term alternative to 3d?" in *Keynote Talk at 3-D Architectures for Semiconductor Integration and Packaging Conference*, 2011.

[4] AMD, *AMD Radeon R9 Fury X Graphics Card. http://support.amd.com/Documents*, 2015.

[5] K. Saban, "Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency," *Xilinx White paper: Vertex-7 FPGAs*, 2011.

[6] NVIDIA, *NVIDIA Tesla P100*, April 2016.

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

[8] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 546–558.

[9] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, "Catnap: energy proportional multiple network-on-chip," in *ACM SIGARCH Computer Architecture News*, vol. 41. ACM, 2013, pp. 320–331.

[10] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*. IEEE, 2008, pp. 203–214.

[11] J. Standard, "High bandwidth memory (hbm) dram," *JESD235*, 2013. [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd235.

[12] J. T. Pawlowski, "Hybrid memory cube: breakthrough dram performance with a fundamentally re-architected dram subsystem," in *Proceedings of the 23rd Hot Chips Symposium*, 2011.

[13] Nvidia, *NVLink, Pascal and Stacked Memory: Feeding the Appetite for Big Data. http://devblogs.nvidia.com/parallelforall*, 2015.

[14] Micron, *Intels Knights Landing Leverages Technology Found in Microns HMC Devices. http://www.micron.com/products/hybrid-memory-cube*, 2015.

[15] M. Li, Q.-A. Zeng, and W.-B. Jone, "Dyxy: a proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 849–852.

[16] S. Ma, N. E. Jerger, and Z. Wang, "Dbar: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 413–424.

[17] R. S. Ramanujam and B. Lin, "A layer-multiplexed 3d on-chip network architecture," *Embedded Systems Letters, IEEE*, vol. 1, no. 2, pp. 50–55, 2009.

[18] C. Liu, L. Zhang, Y. Han, and X. Li, "Vertical interconnects squeezing in symmetric 3d mesh network-on-chip," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. IEEE Press, 2011, pp. 357–362.

[19] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 86–97.

[20] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 86–96.

[21] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: dependency-driven trace-based network-on-chip simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures*. ACM, 2010, pp. 31–36.

[22] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*. IEEE, 2012, pp. 201–210.

[23] B. Black, "Die stacking is happening," in *Intl. Symp. on Microarchitecture, Davis, CA*, 2013.