Scalable Memory Fabric for Silicon Interposer-Based Multi-Core Systems

Itir Akgun*, Jia Zhan*, Yuangang Wang†, and Yuan Xie*
*Department of Electrical and Computer Engineering
University of California, Santa Barbara, California, USA
Email: {iakgun, jzhan, yuanxie} @ ece.ucsb.edu
† Huawei, Shenzhen, Guangdong, China
Email: {wangyuangang} @ huawei.com

***Abstract*—Three-dimensional (3D) integration is considered as a solution to overcome capacity, bandwidth, and performance limitations of memories. However, due to thermal challenges and cost issues, industry embraced 2.5D implementation for integrating die-stacked memories with large-scale designs, which is enabled by silicon interposer technology that integrates processors and multiple modules of 3D-stacked memories in the same package. Previous work has adopted Network-on-Chip (NoC) concepts for the communication fabric of 3D designs, but the design of a scalable processor-memory interconnect for 2.5D integration remains elusive. Therefore, in this work, we first explore different network topologies for integrating CPUs and memories in a silicon interposer-based multi-core system and reveal that simple point-to-point connections cannot reach the full potential of the memory performance due to bandwidth limitations, especially as more and more memory modules are needed to enable emerging applications with high memory capacity and bandwidth demand, such as in-memory computing. To overcome this scaling problem, we propose a memory network design to directly connect all the memory modules, utilizing the existing routing resource of silicon interposers in 2.5D designs. Observing the unique network traffic in our design, we present a design space exploration that evaluates network topologies and routing algorithms, taking process node and interposer technology design decisions into account. We implement an event-driven simulator to evaluate our proposed memory network in silicon interposer (MemNiSI) design with synthetic traffic as well as real in-memory computing workloads. Our experimental results show that compared to baseline designs, MemNiSI topology reduces the average packet latency by up to 15.3% and Choose Fastest Path (CFP) algorithm further reduces by up to 8.0%. Our scheme can utilize the potential of integrated stacked memory effectively while providing better scalability and infrastructure for large-scale silicon interposer-based 2.5D designs.**

I. Introduction

Three-dimensional (3D) integration is a promising solution to overcome the memory wall problem industry faces nowadays. 3D-stacked memory such as High Bandwidth Memory (HBM) [1] provides larger memory capacity due to increased density, higher bandwidth due to Thermal Silicon Via (TSV) 3D integration technology, lower power consumption due to reduced interconnect, and better performance due to the integration of memory closer to the processor. Therefore, die-stacked memory is considered as a promising solution to enable applications, such as in-memory computing and High Performance Computing (HPC), that require high memory density, bandwidth and capacity.

This work was supported in part by the National Science Foundation under grants 1500848 and 1533933.

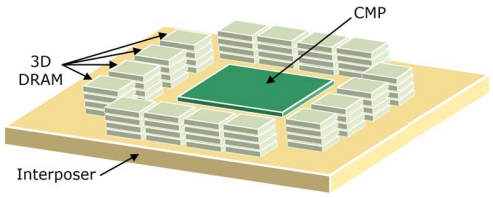

Fig. 1: Silicon interposer-based 2.5D design

Although memory technologies have matured enough to exploit 3D integration, stacking them on top of the processor chip exacerbates the thermal problems due to increased power density and therefore requires costly thermal solutions [2]. On the other hand, 2.5D implementation of integrating processors and 3D-stacked memory on silicon interposer, as shown in Figure 1, results in a lower power density and is a more cost-efficient option to enable such large-scale designs [3]. Industry confirms this trend with the emergence of 2.5D products such as AMD's Radeon 9 Series Fury X [4], [5] and NVIDIA's Pascal [6]. Furthermore, unlike 3D integration where the area of the processor is a limiting factor for the stacked memory capacity, the disaggragate nature of 2.5D integration enables integration of more on-package memory and better scalability.

Previous work [7] has adopted Network-on-Chip (NoC) concepts as the communication fabric for 3D integration of CMP systems with stacked and distributed L2 caches, which improves the performance by reducing the access latency by moving from 2D to 3D design and replacing long interconnects with vertical interconnects. With the advent of 2.5D designs, the notion of a hybrid network has emerged, which exploits the already-available resources of silicon interposer [8], [9], [10] to implement interconnect in silicon interposer, along with the NoC in the CPU die, so that CPU-to-CPU traffic and CPU-to-memory traffic can be directed into two separate networks. However, designing a hybrid network for a 2.5D design has its unique challenges compared to 3D. First of all, 2.5D is more bandwidth limited as the microbumps which provide connectivity between the processor chip and the silicon interposer are limited by the perimeter of the chip [10].

978-1-5090-5142-7/16/$31.00 ©2016 IEEE

As more memory modules are integrated, the per-memory stack bandwidth decreases even further since more modules would share the bandwidth as the total number of microbump connections of the CPU chip is fixed. Second, the layout of the memory modules on a silicon interposer is constrained by the CPU chip. The way memory and CPU modules are connected can therefore result in long wires. Lastly, silicon interposer can be manufactured in a different process technology than the CPU chip for cost benefits [10], which in turn would create a difference in maximum attainable frequency. Therefore, all of these constraints should be taken into account when designing an interconnection network for 2.5D designs.

Scalability is one major concern in large-scale system designs. For DIMM-based memory systems, as the data rate of DRAM increases, the number of DIMMs that can be supported on a conventional multi-drop bus decreases. Alternatively, multiple DIMMs can be connecting via point-to-point links [11], rings [12], or trees [13], [14], but all suffer from latency or power overhead, or are limited to specific RF interconnect or silicon photonics technologies. For 2.5D systems, Jerger et al. [8] propose using the already-available resource of the silicon interposers to implement a "duplicated" network for supplementing NoC bandwidth, but still deploys point-to-point connections for CPU-to-memory communication. However, as the need for integration of more and more memory modules increases due to workload demand of such applications as in-memory computing [15], [16], [17], such a design fails to support efficient integration of more modules while keeping the performance up to par. Recently, Kim et al. [18], [19] has proposed a memory-centric architecture for connecting multiple hybrid memory modules (HMCs) in a multi-socket server system, based on 3D stacking technologies. Zhan et al. [20] demonstrated how an inter-memory network helps provide scalability and proposed co-optimization of inter- and intra-memory network for in-memory computing. However, there is still a lack of study on integrating multiple memory modules based on silicon interposer technologies.

In this work, we propose a memory-centric architecture based on silicon interposer to achieve high bandwidth and low latency in 2.5D integrations, taking advantage of the existing routing resources in order to better utilize the available memory performance. Observing the unique network traffic in 2.5D designs, we present a design space exploration that evaluates network design, topology, and routing algorithms, while taking process node and interposer technology design decisions into account. We implement an event-driven simulator and evaluate our processor-memory network designs with both synthetic traffic and real in-memory computing workloads. Experimental results presented in Section V-C show that proposed Memory Network in Silicon Interposer (MemNiSI) topology reduces the average packet latency by up to 15.3% and Choose Fastest Path (CFP) algorithm further reduces by up to 8.0% compared to baseline designs. Our scheme better utilizes the potential of integrated stacked memory to provide improved scalability and infrastructure for large-scale silicon interposer-based 2.5D designs.

II. Background

In this section, we introduce the silicon interposer technology which enables 2.5D integration. Then, we describe our target applications: in-memory computing, which drives the design of scalable memory systems.

A. Silicon Interposer

An interposer is a silicon die with metal layers on top that allows for face-down integration of chips as shown in Figure 1. Micro-bumps sit between the metal layers of the silicon interposer and the chip, providing electrical connection. Micro-bumps can be manufactured with a pitch as small as 20 μm [21]. The bandwidth of the chip on silicon interposer therefore can be calculated given the micro-bump pitch and the perimeter of the stacked chip for interposer-based designs [10]. Furthermore, the wire characteristics of interposer metal layers are similar to on-chip interconnects with negligible impedance from micro-bumps [9].

Unlike 3D-stacked memory whose capacity is limited by the size of the processor chip, more memory nodes can be integrated in 2.5D systems as long as there is sufficient space on the silicon interposer. However, 3D stacking potentially provides more memory bandwidth because the number of TSV connections is basically proportional to the surface area of the processor chip, while the bandwidth between 2.5D-stacked chips is bounded by their perimeters. However, considering the additional area of TSVs, the bandwidth gap between 3D and 2.5D implementation is significantly reduced, and both technologies will provide substantial bandwidth improvement over traditional DIMM-based DRAM systems. Consider a large-scale interposer-based system similar to NVIDIA's Pascal or AMD's Fury X, with a processor die of size 600 mm^2. Assuming a square die and a microbump pitch of 20 μm, the die can have 4,900 microbumps in 2.5D implementation. Letting each signal operate at 1 Gbps, we get a total bandwidth of 612 GB/s. On the other hand, a 3D integration of the same die would result in 30,747 connections using 50 μm TSV pitch, reaching 3.8 TB/s of total bandwidth.

There are two implementation types of silicon interposers: passive and active. Passive interposer consists only of the metal layers and is the one that is manufacturable in the near future. Passive interposer integration can provide interconnection in the silicon layer. One downside of using passive interposer is the inability to implement repeaters on long wires to increase the interconnect speed. On the other hand, active interposer implements active devices in the interposer along with routing capability. Active interposers pose an extra cost challenge over passive interposers due to active devices increasing the critical area and therefore leading to lower interposer yields. A way to achieve cost benefits is to implement the interposer in an older technology to achieve high yield.

B. In-Memory Computing

In-memory computing has become a widely adopted technology for real-time analytics applications in the industry. One important application is in-memory databases such as SAP

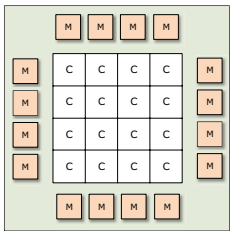

Fig. 2: Physical layout of the 2.5D design

HANA [15] and IBM DB2 BLU [22], which strive to *fit an entire database in memory*, potentially obviating the need of paging data to/from mechanical disks. The open-source community is also actively advancing big data processing frameworks that leverage in-memory computing. For example, Spark [16] is a cluster computing framework optimized for data reuse in iterative machine learning algorithms and interactive data analysis tools. Unlike MapReduce [23] which writes data into external storage system for data reuse between computations, Spark creates a new distributed memory abstraction and lets users explicitly *persist intermediate datasets in memory*, which can achieve up to 100× performance speedup [16].

As we can see, in-memory computing relies heavily on the memory capacity to keep the application's entire working dataset in memory for faster access. Therefore, we claim in-memory computing as one of the promising applications which can leverage a large-scale interposer-based system with significant in-package memory capacity and scalability.

III. Target Architecture

The main goal of this work is to build a scalable interposer-based network architecture for integrating a large number of memory modules in a multi-core system, in order to enable emerging applications with high memory capacity and bandwidth demand, such as in-memory computing. Current commercial products integrate four memory stacks on a passive silicon interposer, connected to the CPU die with point-to-point links [5]. To study scalability, our model assumes a CPU chip and 16 individual die-stacked memory modules, integrated on a silicon interposer. The CPU chip consists of a chip multiprocessor (CMP) with 16 cores, connected via a Network-on-Chip (NoC) in a 2D mesh topology. Surrounding the CPU chip, there are 16 individual die-stacked memory modules (such as HBM), as shown in Figure 2. In this physical layout, we do not specify the how the memory modules are connected to the CPU chip as we will explore potential network topologies in Section IV-A. CPU and memory modules are integrated on a silicon interposer and connected to the metal layers of the silicon interposer via microbumps. The unique processor-memory traffic goes through the corner cores and corner memory modules—pillar nodes—creating a potential bottleneck at these routers. Therefore, in Section IV we discuss optimizing the topology, router design, and routing algorithm to mitigate this bottleneck.

IV. Network Design

While there exists multiple studies on NoC design in the CPU die, our focus is on the memory fabric design using the already available resources in silicon interposer. In this section, we will discuss the detailed network architecture design to integrate a large number of memory modules on the same silicon interposer-based system.

A. Topology

In this section, we consider three memory fabric topologies that provide connection between the CPU chip and memory modules using the available resources in silicon interposers for routing capabilities. All these topologies are implemented on the same physical layout shown in Figure 2, the only difference between them being the interconnections. The first design, as illustrated in Figure 3a, implements point-to-point connections between the memory modules and the CPU cores along the edges of the chip. Although each of the memory modules is only one hop away from its corresponding pillar CPU node, they all share the bandwidth of the CPU chip due to the fixed number of micro-bumps. As a result, the total memory bandwidth is statically partitioned to each memory module, limiting the available per-memory bandwidth. This problem will exacerbate as the number of memory modes increases.

The second baseline design, as shown in Figure 3b, quadruples the per-memory bandwidth compared to point-to-point design by connecting only four of the memory modules to the CPU chip and the rest to these pillar modules with a daisy chain. However, with this implementation, the number of hops to reach the memory modules is increased due to packets traversing the daisy chain and congestion might occur at the pillar nodes. Moreover, the total memory address space is still partitioned into four independent regions, causing inefficiency in memory sharing. For example, if the dataset of an application resides in two memory modules separated into two diagonal regions, accesses between CPU and memory may have to traverse the entire NoC, causing long end-to-end delay.

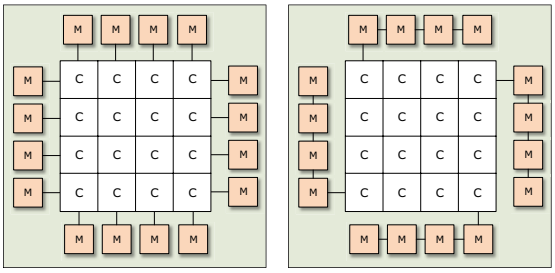

Fig. 3: Baseline network topologies (a) Point-to-point, and (b) Daisy chain.

Notice that the prior two designs only use the interposer for edge-to-edge communication between adjacent chips (e.g. from CPU to stacked DRAMs). However, apart from the routing resources along the edge of the interposer, a vast

majority of the interposer's area and routing resources are unused. Therefore, we leverage the otherwise wasted routing resources of the interposer to implement CPU-to-memory connections. Prior work [8] has proposed similar concepts but only has used the interposer to implement a duplicated NoC. In their case, CPU-to-CPU (cache coherence) traffic will use the on-chip network, while the CPU-to-memory traffic will use the interposer network. While this design reduces interference of these two traffic classes, the benefit is limited because the NoC bisection bandwidth is already large enough to accommodate the traffic of both. For example, given 128-bit NoC links operating at the same frequency as CPU (3 GHz), a 4×4 mesh NoC would provide 384 GB/s bisection bandwidth.

Therefore, instead of using the interposer routing resources for only CPU-to-memory connections, we implement a memory network in the interposer to interconnect all the memory modules directly, forming a memory-centric architecture. Figure 4 shows the logical layout of our proposed memory network in silicon interposer (*MemNiSI*). Note that this scheme occupies the same area as the previous designs physically; however, the illustration depicts the logical connections for clarity.

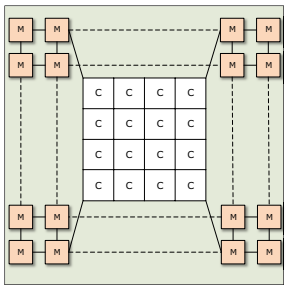

Fig. 4: Logical layout of the proposed memory network topology, MemNiSI.

As we can see from this architecture, all of the memory nodes are connected using a mesh network. Even though memories do not directly communicate with each other, the memory network provides additional routing flexibility which potentially reduces the end-to-end communication distances, without sacrificing the per-memory bandwidth. A detailed analysis of the routing algorithms will be provided in Section IV-C. In the meantime, a drawback of this design is the existance of longer wires connecting the memory modules across the CPU chip.

Since none of these designs is a clear winner, in Section V-C1 we evaluate all three topologies with the routing algorithms introduced in Section IV-C under both synthetic traffic and real workloads.

B. Technology

Apart from the topology choices as discussed above, there are two important factors that influence the memory fabric design in silicon interposer-based systems: interposer type and process technology.

Silicon interposers can be either active or passive, meaning they can incorporate active devices or not, respectively, as described in Section II-A. Our scheme is applicable to both of these design choices. With active interposer, the routers for the memory network can be implemented in the interposer itself. On the other hand, for passive interposer implementation, we assume the routers are implemented in a logic base under the stacked memory modules so that only wires go through the silicon interposer. Note that the choice between active and passive interposer does not affect the logical network topology. However, taking process technology into consideration, there are several variances that influences our design, as will be discussed next.

There are three outcomes to consider for the various process technology design decisions. The first case assumes the NoC and network in silicon interposer run at the same frequency. This is a valid assumption because the silicon interposer interconnect speed can be as fast as on-chip communication [8]. On the other hand, the second case considers that the interposer is passive and is implemented with the same process technology as the processor chip. Since repeaters can be used for long wires within the CPU chip to increase the interconnect frequency but not in the passive interposer, NoC will be faster than network in silicon interposer. Finally, network in silicon interposer can be faster than NoC if it is manufactured with an older technology. Using an older technology has two advantages in this design: cost benefits due to higher yield and faster interconnect. The trends in process technology show that interconnect scaling exacerbates as we move to newer process nodes [24]. With these factors to consider, the actual frequency discrepancy between the silicon interposer network and the on-chip network may vary from technology to technology. Therefore, in Section V-C2, we provide a sensitivity analysis of our network design by sweeping the frequencies of the two networks.

C. Routing Algorithm

As mentioned in the previous topology design, memory network provides additional routing flexibilities for end-to-end memory accesses. In this section, we discuss the corresponding routing algorithms. Note that different topologies can be applied to our memory network design, such as torus, flattened-butterfly, etc. We use a generic mesh topology as a case study, while various topologies will cause different design tradeoffs and the search for the ideal network topology in silicon interposer is out of the scope of this paper. Therefore, we can simply adopt dimension ordered routing which provides shortest distance routing while guaranteeing to be deadlock-free.

1) Baseline Designs: Point-to-point and daisy chain topologies implement a variety of XY routing algorithms that we call pillar router first (PRF) routing. PRF routing algorithm first routes the packets to the pillar router closest to the memory module using X-first Y-last routing algorithm and then to the destination router, again using X-first Y-last routing.

2) Memory Network in Silicon Interposer (MemNiSI) Design: Memory network topology allows more routing algorithms to be implemented. For CPU to memory traffic, the flow

path is to route a packet from CPU to a pillar node through the NoC and then traverse the memory network to the destination. Reversely, for memory responses, the packets will traverse the memory network before being sent back on chip, and follow the NoC to reach the destined CPU. However, there are several ways to implement this algorithm as discussed below, all based on the XY routing algorithm.

Network in silicon interposer heavy (NiSIH) routing algorithm aims to utilize the silicon interposer more heavily by routing the packets to spend more hops in the silicon interposer network. For CPU to memory traffic, NiSIH first routes to the *closest pillar router to the CPU node* and from the pillar router to destination memory using XY routing via silicon interposer network. For memory to CPU traffic, however, NiSIH routes packets via silicon interposer network back to the *closest pillar router to the CPU* and from the pillar router to destination CPU using XY routing via NoC. Network-on-chip heavy (NoCH) routing algorithm, in contrast to NiSIH algorithm, utilizes the network-on-chip more heavily. CPU to memory traffic is routed via the *closest pillar router to the memory node* through NoC first, using XY routing. Similarly, NoCH routes the memory to CPU traffic via the silicon interposer network to the *closest pillar router to the memory node*, and from the pillar node to the destination CPU via NoC. Figure 5 illustrates the NiSIH and NoCH routing algorithms. We expect NoCH routing algorithm to perform better than NiSIH in designs where NoC is faster and NiSIH to perform better than NoCH in designs where the silicon interposer network is faster.

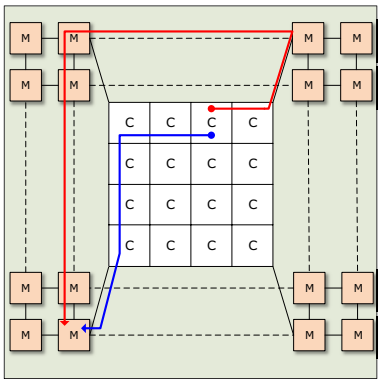

Fig. 5: Illustration of NiSIH and NoCH routing algorithms on memory network topology, shown in red and blue respectively. Destination node is six hops away from the source node using either algorithm. Memory access response messages follow the corresponding request paths back.

Finally, considering the frequency discrepancy of on-chip network and silicon interposer network, we propose Choose-Faster-Path (CFP) routing algorithm, as shown in Algorithm 1, which estimates the time it takes from source to destination with NiSIH and NoCH given the number of hops from source to destination router and the network frequencies. Specifically, CFP will dynamically select the routing path that yields the fastest expected route for each instance. Note that we can further improve the accuracy of our dynamic routing path selection process in a number of ways. In practice, not all the links between memory modules may be of the same length. CPF can be enhanced to take the heterogeneity between the link latencies into account while evaluating the fastest path. The exact implementation is dependent on the specific layout and topology considerations. Another improvement to CPF may leverage the congestion status of different routes. However, implementing this will incur significant overhead due to introduction of additional counters and synchronization logic to track the utilization and delay of individual routers. We leave it as future work to analyze the design trade-offs.

Algorithm 1 Choose Faster Path (CFP) Algorithm

Input: Current node, destination node
Output: Routing path
/* Determine pillar routers */
$close\_pillar$:= closest pillar to current node
far_pillar := closest pillar to destination node
/* Calculate # of hops to destination */
$dest\_close\_NoC$:= # hops in NoC to/from $close\_pillar$
$dest\_close\_NiSI$:= # hops in NiSI to/from $close\_pillar$
$dest\_far\_NoC$:= # hops in NoC to/from far_pillar
$dest\_far\_NiSI$:= # hops in NiSI to/from far_pillar
/* Calculate total expected time to destination */
$total\_close = (dest\_close\_NoC \times NoC\_latency) + (dest\_close\_NiSI \times NiSI\_latency)$
$total\_far = (dest\_far\_NoC \times NoC\_latency) + (dest\_far\_NiSI \times NiSI\_latency)$

if $total\_close \leq total\_far$ **then**
  Route via $close\_pillar$
else
  Route via far_pillar
end if

V. Experiments

In this section, we first describe the infrastructure and workloads setup to evaluate the 2.5D processor-memory interconnect design choices. Next, we present our results for topology study, sensitivity analysis, and routing algorithm study to determine the most optimal design for scalable interposer-based designs.

A. Simulator Setup

We implemented an event-driven simulator in C++ to model the hybrid network of the interposer-based design introduced in Section III. The simulator can model an interconnect network that allows for specification of network design details such as the topology and routing algorithms. Each CPU core is implemented with an active router which can issue memory and cache coherence requests according to the traffic pattern. The individual die-stacked memory modules share the total available in-package memory capacity and are implemented with a passive router which can only receive and reply to the memory access requests issued by CPU cores. As explained in Section III, the hybrid network consists of a 2D mesh network-on-chip that connects the CPU cores together and a network in silicon interposer that connects the memory modules to the processor chip via silicon interposer. We consider the

design choices of network topology and routing algorithm, and distinct link frequencies for the network in silicon interposer. Table I shows the basic configurations of CPU, memory, and interconnect.

B. Workloads Setup

In our simulator we implement two synthetic traffic patterns, as described in Table II, *uniform-random* and *hotspot*, with the option of additional CPU-to-CPU communication such as cache coherence to stress the NoC. *Uniform-random* traffic issues memory requests that are uniformly distributed to all memory nodes. *Hotspot* traffic can issue a selected percentage of memory requests to one specified memory node to create a hotspot.

In order to evaluate real workloads, we extend our simulator to work with memory traces. We consider the following in-memory computing workloads and collect their instruction and memory traces use a Pin-based [28] functional simulator on a Dell PowerEdge T630 server: Pagerank and Memcached from CloudSuite benchmark [25], Redis benchmark [27], and Spark-Wordcount, Spark-Grep, and Spark-Sort from Wikipedia dataset provided in BigDataBench benchmark [26], using Spark-1.4.1 [29]. A detailed description of these workloads is provided in Table III.

C. Results

In this section, we perform sensitivity analysis and study topology and routing algorithm choices for the processor-memory network in 2.5D designs using the event-driven simulator described in previous sections under both synthetic traffic and real in-memory computing workload traces.

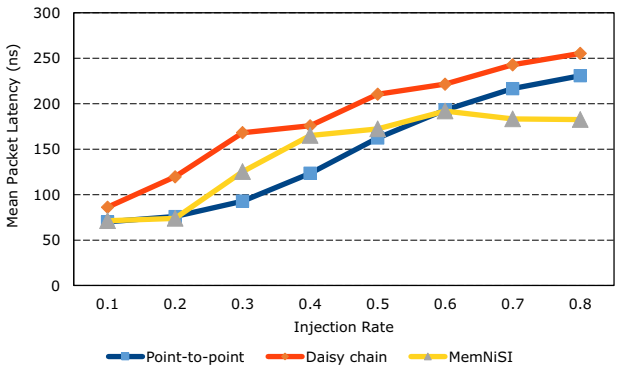

Fig. 6: Average packet latency comparison of the network topologies under *uniform-random* traffic.

1) Topology Study: We assume equal network frequencies while studying network topologies. PRF routing algorithm is used in point-to-point and daisy chain designs, whereas NiSIH routing algorithm is used in MemNiSI design. Figure 6 shows the mean packet latency of the network topologies with respect to the injection rate under *uniform-random* traffic. Point-to-point topology performs the best under low network load as each memory node is only one hop away from the NoC. Daisy chain, on the other hand, is the slowest topology due to the longer average distance between CPU and memory nodes and the congestion along the daisy chains. Under low network load, memory network design MemNiSI performs in between the two baseline designs as the average distance between the source and destination pairs heavily determine the packet latency. MemNiSI outperforms both designs under heavy network load since the network is not as heavily congested as memory network helps disperse the requests. Furthermore, we observe that while the average packet latency of point-to-point and daisy chain designs steadily increase as the network load increases, MemNiSI levels out, suggesting that it is a more scalable design under high network demand.

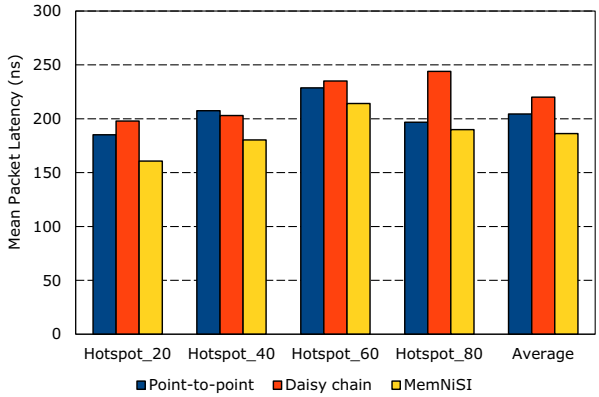

Fig. 7: Average packet latency comparison of the network topologies under *hotspot* traffic. Lower is better.

Figure 7 compares the average packet latency of the three network topologies under *hotspot* traffic which sends the specified percentage of traffic to a single memory node. On average, point-to-point performs slower than MemNiSI design since MemNiSI performs better under heavy network load of hotspot traffic. Daisy chain topology proves to be the least scalable design as the packet latency increases steadily with the hotspot traffic. MemNiSI is the fastest and most scalable design by being 8.92% and 15.33% faster compared to point-to-point and daisy chain topologies under *hotspot* traffic, on average.

2) Sensitivity Analysis: In order to evaluate our proposed network design better, we perform sensitivity analysis on relative network-on-chip and network in silicon interposer frequencies.

Figure 8 shows the network frequency sensitivity analysis under synthetic *uniform-random* traffic for CFP algorithm. As introduced in Section IV-C, CFP routing algorithm accounts for the number of hops between source and destination pairs for NiSIH and NoCH algorithms, and picks the one that would yield the fastest route, taking the network frequencies into account. By changing the network frequency ratio of network-on-chip to network in silicon interposer, we are able to observe the algorithm that is being favored. We can observe that for the configurations that NoC is faster than network in silicon interposer (NiSI), NoCH algorithm is favored over NiSIH, and vice versa, since it is faster overall if the packets are routed via the faster network for most of the distance. Another

TABLE I: System (CPU, memory, and interconnect) configurations

CPU & Memory	16 cores, 2GHz; 64B cache-line size; 16 memory nodes; 4GB per node; 100 cycles memory latency
On-chip network	4-stage router pipeline; 4 VCs per port; 4 buffers per VC; flit size = 16B; maximum packet size = 4 flits
Baseline interposer network	4×4 mesh topology, same router configurations as the on-chip network.

TABLE II: Different synthetic traffic patterns

Uniform-Random	Every processor sends an equal amount of traffic to every memory node.
Hotspot	Every processor sends a large portion (e.g. 50%) of traffic to a single memory node. The rest of the traffic distribution is uniform-random.

TABLE III: Real in-memory computing workload characteristics

Pagerank	From the graph analysis benchmark in CloudSuite [25], which runs PageRank on a Twitter dataset with 11M vertices. GraphX is used to run PageRank in Spark.
Tunkrank	A measure of Twitter influence. From the graph analysis benchmark in CloudSuite [25]. Twitter dataset with 11M vertices. Data set size 1.3GB, and GraphLab requires 4GB heap memory.
Spark-grep	A "grep" job running on Spark, which extracts matching strings from text files and counts how many times they occurred with the Wikipedia dataset provide in BigDataBench [26].
Spark-sort	A "sort" job running on Spark, which sorts values by keys with the Wikipedia dataset provide in BigDataBench [26].
Memcached	From CloudSuite [25], which simulates the behavior of a Twitter caching server using the Twitter dataset with 8 client threads, 200 TCP/IP connections, and a get/set ratio of 0.8.
Redis	An in-memory database system which simulates running 50 clients at the same time sending 100,000 total queries [27].

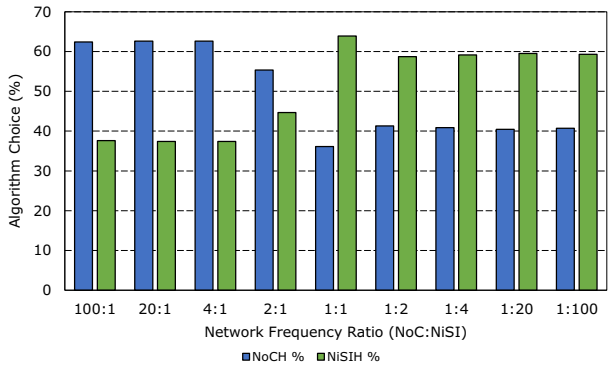

Fig. 8: Sensitivity analysis for Choose Faster Path (CFP) algorithm which shows the algorithm choice between NiSIH and NoCH for various configurations of network frequency ratios of network-on-chip and network in silicon interposer.

observation is that the algorithm choice difference stabilizes quickly and how much faster one of the networks is, the packet latency is bounded by the other and so the relative picks of the two algorithms remain the same. Therefore, we pick 4:1 configuration for when NoC is faster and 1:4 for when silicon interposer network is faster for our following experiments.

3) Routing Algorithm Study: In this section, we evaluate our MemNiSI topology using three routing algorithms: NiSiH, NoCH, and CFP under synthetic traffic and real in-memory workloads.

Figure 9 provides a routing algorithm comparison under synthetic traffic for the three network frequency cases, normalized to NiSIH routing algorithm separately for each traffic. For the case where NoC and NiSI are equally as fast, NiSIH performs better than NoCH as also observed in Figure 8. This is due to the fact that memory network in silicon interposer reduces the average distance between CPU and memory nodes. For the cases where the two network frequencies differ, the algorithm that leverages the faster network more yields the lower packet latency as expected. CFP always performs close to or better than the better performing one between NiSIH and NoCH in the particular case. The reason why CFP can perform better is that algorithm pick between NiSIH and NoCH is performed for every source-destination pair, greedily choosing the fastest for each packet. CFP can outperform up to 6.85%.

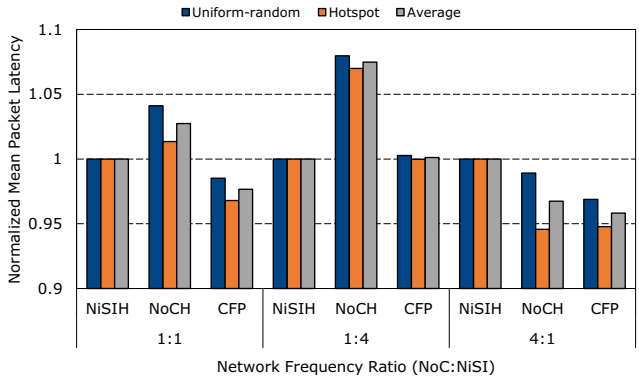

Fig. 9: Routing algorithm comparison under *uniform-random* and *hotspot* traffic, normalized to NiSIH algorithm. Results show the various configurations of network frequency ratios of network-on-chip and network in silicon interposer.

Figure 10 compares routing algorithms for the in-memory computing workloads specified in Section V-B. The results are normalized to NiSIH routing algorithm. On average, CFP outperforms NiSIH by up to 3.38% and NoCH by up to 10.03%. For the case where the frequencies of the two networks are equal, CPF performs better on average than NiSIH by 1.65% and NoCH by 7.99%. This shows that memory network in silicon interposer designs will benefit even from a naive CFP implementation.

Experimental results show that a memory network approach suits well to scalable interposer-based designs. In this work, we mainly focus on providing design considerations and are only able to explore a small part of the vast design space. Even though this paper focuses on evaluating the network latency of a system of scalable memory, we plan to consider other metrics, such as power, area, cost, and reliability in order to compare processor-memory network designs in our future work.

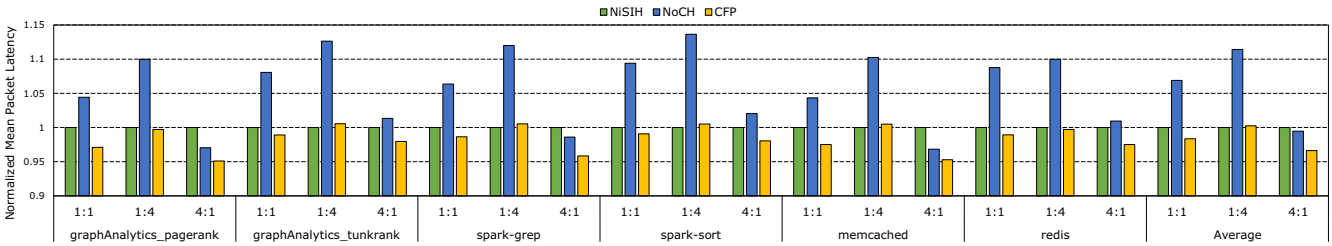

Fig. 10: Routing algorithm comparison under real in-memory computing workloads for NoC to NiSI frequency ratios of 1:1, 1:4, and 4:1, normalized to NiSIH. Results show the various configurations of network frequency ratios of network-on-chip and network in silicon interposer.

VI. Conclusion

One of the major challenges in large-scale interposer-based 2.5D designs is providing a scalable fabric for high-bandwidth, low-latency communication between the processor chip and stacked memory nodes. In this work, we demonstrate that the existing processor-memory network of 2.5D designs cannot provide the full potential of the memory performance and memory-centric network design in silicon interposer is a promising scalable solution. We present a design space exploration that evaluates network design, topology, and routing algorithms, taking process node and interposer technology design decisions into account. Our scheme achieves better utilization of integrated stacked memory while allowing better scalability and infrastructure for large-scale silicon interposer-based 2.5D designs.

References

[1] J. Kim and Y. Kim, "Hbm: Memory solution for bandwidth-hungry processors," in *Hot Chips*, vol. 26, 2014.

[2] X. Wu *et al.*, "Thermal-aware 3d ic designs," *3D Integration for VLSI Systems*, p. 313, 2011.

[3] D. Stow *et al.*, "Cost and Thermal Analysis of High-Performance 2.5D and 3D Integrated Circuit Design Space," in *2016 IEEE Computer Society Annual Symposium on VLSI*, July 2016.

[4] "AMD Radeon R9 Series Graphics Cards with High-Bandwidth Memory," http://www.amd.com/en-us/products/graphics/desktop/r9.

[5] J. Macri *et al.*, "AMD's next Generation GPU and Memory Architecture," in *Hot Chips*, 2015.

[6] "NVLink, Pascal and Stacked Memory: Feeding the Appetite for Big Data," http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/.

[7] F. Li *et al.*, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," in *33rd International Symposium on Computer Architecture (ISCA)*, 2006, pp. 130–141.

[8] N. E. Jerger *et al.*, "NoC Architectures for Silicon Interposer Systems: Why Pay for more Wires when you Can Get them (from your interposer) for Free?" in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 458–470.

[9] A. Kannan *et al.*, "Enabling Interposer-based Disintegration of Multi-core Processors," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: ACM, 2015, pp. 546–558. [Online]. Available: http://doi.acm.org/10.1145/2830772.2830808

[10] G. H. Loh *et al.*, "Interconnect-Memory Challenges for Multi-chip, Silicon Interposer Systems," in *Proceedings of the 2015 International Symposium on Memory Systems*, ser. MEMSYS '15. New York, NY, USA: ACM, 2015, pp. 3–10. [Online]. Available: http://doi.acm.org/10.1145/2818950.2818951

[11] B. Ganesh *et al.*, "Fully-buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling," in *IEEE 13th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2007, pp. 109–120.

[12] A. N. Udipi *et al.*, "Combining Memory and a Controller with Photonics Through 3D-stacking to Enable Scalable and Energy-efficient Systems," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2011, pp. 425–436.

[13] K. Therdsteerasukdi *et al.*, "The DIMM tree architecture: A high bandwidth and scalable memory system," in *IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 388–395.

[14] T. J. Ham *et al.*, "Disintegrated Control for Energy-efficient and Heterogeneous Memory Systems," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 424–435.

[15] F. Färber *et al.*, "SAP HANA database: data management for modern business applications," *ACM Sigmod Record*, vol. 40, no. 4, pp. 45–51, 2012.

[16] M. Zaharia *et al.*, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[17] A. Daglis *et al.*, "Manycore Network Interfaces for In-Memory Rack-Scale Computing," in *ACM/IEEE 42nd International Symposium on Computer Architecture (ISCA)*. ACM, 2015, pp. 567–579.

[18] G. Kim *et al.*, "Memory-centric system interconnect design with Hybrid Memory Cubes," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, Sept 2013, pp. 145–155.

[19] ——, "Multi-gpu system design with memory networks," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 484–495.

[20] J. Zhan *et al.*, "A Unified Memory Network Architecture for In-Memory Computing in Commodity Servers," in *Proceedings of the 49th ACM/IEEE International Symposum on Microarchitecture (MICRO)*, 2016.

[21] R. Huemoeller, "Through Silicon Via (TSV) Product Technology," Amkor Technology, Technical Report. Presented to IMAPS North Carolina Chapter, 2012.

[22] V. Raman *et al.*, "DB2 with BLU acceleration: So much more than just a column store," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1080–1091, 2013.

[23] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[24] R. Ho, K. Mai, and M. Horowitz, "Managing wire scaling: a circuit perspective," in *Interconnect Technology Conference, 2003. Proceedings of the IEEE 2003 International*, June 2003, pp. 177–179.

[25] M. Ferdman *et al.*, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 37–48, 2012.

[26] L. Wang *et al.*, "Bigdatabench: a big data benchmark suite from internet services," in *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 488–499.

[27] "Redis Benchmark," http://redis.io/topics/benchmarks.

[28] H. Patil *et al.*, "Pinpointing representative portions of large Intel® Itanium® programs with dynamic instrumentation," in *IEEE/ACM 37th International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2004, pp. 81–92.

[29] "Spark 1.4.1," http://spark.apache.org/downloads.html.