COMMUNICATION AND COMPUTATION OPTIMIZATIONS IN MODULAR SYSTEMS-ON-CHIP

A Dissertation

by

PRITAM MAJUMDER

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Eun Jung Kim
Committee Members,	Daniel A. Jiménez
	Paul Gratz
	Duncan M. "Hank" Walker
	Abdullah Muzahid
Head of Department,	Scott Schaefer

May 2022

Major Subject: Computer Engineering

ABSTRACT

In our modern world where everyone is always connected through internet, terabytes of data gets generated at every moment through online activities like communication on social media, online banking, online shopping, browsing, streaming, telemedicine, information on global activities, weather forecasting, astronomy etc. Current e-commerce and science community are heavily dependent on this data, creating huge demand for quick data processing through machine learning methods for on time decision making and also for enhancing our knowledge regarding science and universe. To meet the demand, currently we rely on scale-out systems like cloud servers, usually equipped with GPUs as general purpose accelerators, often realised with SoCs.

For designing large scale systems, design modularity offers cheaper SoC with inherent integration complexity, like network deadlock involving multiple modules, in spite of each individual module being deadlock-free. Based on our first observation, the deadlock in modular SoC is formed by forming circular channel dependency involving two or more modules in the SoC. Further, mixing traffics originated from different modules may block each other in a circular fashion, which results in deadlock. We propose a deadlock avoidance technique for any modular SoC to make the integration deadlock free with minimum overhead. We evaluate our theory of deadlock freedom using full-system SoC simulation constituted of independently designed CPUs and GPUs through interposer network. The routing used in CPU, GPU, or in interposer are completely independent of each other, experience deadlock while exposed to high workload. Our technique successfully avoids the deadlock with much lesser performance and energy overhead than the state-of-the-art turn-restriction-based SoC deadlock solution.

In addition, the excessive data movement in these large systems, tackled by near-memory processing (NMP) has further scope for improvement in their data and computation mapping, as reducing data movement does not ensure optimized operation cost. We propose a reinforcement learning (RL) based solution to improve the cost of operations by improving the data and computation mapping in the memory network for NMP. The solution constituted of two main components,

(1) the formulation of the mapping optimization as an RL problem that involves selecting enough information from the system to form states, deciding the actions based on the desired mapping outcome, and properly calibrate feedback aiming towards the goal of performance improvement, (2) the realization of the information collection system and efficient implementation of data and computation remapping. We integrate RL framework with system simulation framework and show that our technique added as a plug-in module in the system can significantly improve the performance of the NMP techniques. Finally we describe the simulation framework in details, integrated with reinforcement learning, which we develop from scratch to evaluate our proposed solution.

DEDICATION

To my loving family

To the past, the current and the future

ACKNOWLEDGMENTS

Without the support of many people, this dissertation would not have been possible. First and foremost, I would like to express my heartfelt gratitude to my advisor, Dr Eun Jung (EJ) Kim. Since the first day, her continuous encouragement has inspired me to go through the ups and downs along this journey. Her inspiring guidance, generous support and continuous trust has motivated new directions and perspectives in interconnection networks, leading to the success of this research. In addition to her technical expertise, Dr Kim has also offered me advice on my pursuit of career goals, I am truly indebted to her. Many thanks to my committee members and collaborators. I am thankful to Prof. Ki Hwan Yum for his suggestions and efforts in shaping this research. I am grateful to Dr Daniel A. Jiménez, Dr Paul Gratz and Dr Abdullah Muzahid for the insightful feedback and the mentoring they have provided for my research and career development. I would also like to thank Dr Chia-Che Tsai for wonderful collaboration and support.

It would not have been a wonderful graduate life without friends. First, thanks to my HPC research group mates, especially Jiayi and Sungkeun who have made my (office) life very colorful through chit-chat, debates, jokes, playing foosball, badminton, table-tennis, tennis and occasionally cooking together. I also appreciate the efforts that Richa, Ram, Divya, Swathi and Dylan have contributed to this research. I would also like to thank Farabi, Harpreet, Sunyoung for wonderful collaboration. Thanks to Sanchali for supporting being a part of my life. Special thanks to Esha and Pinaki for being such a wonderful friend. Thank you Abhishek, Shudipta, Herede, Hia for being so amazing friends from childhood. Special thanks to Somsankar as a friend for boosting my confidence during IIT Madras entrance tests. Thank you Riddhi for being such a good apartment mate for majority period of my PhD. I am also grateful to the teachers and mentors who have been guiding and supporting me along my education journey. Finally I thank all my family members, especially my parents and my sister for supporting me in all my ups and downs and always having faith in me.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Dr Eun Jung Kim (advisor), Dr Daniel A. Jiménez, and Dr Duncan M. "Hank" Walker, Dr Abdullah Muzahid of the Department of Computer Science and Engineering, and Dr Paul Gratz of the Department of Electrical and Computer Engineering.

In Chapter 3 the hardware synthesis is helped by Divya and Swathi as Masters student in Texas A&M University and part of it has been published in IEEE Transaction on Computers on 2021. In Chapter 4 in concept development Jiayi helped. In Chapter 5 in some of the functional parts SungKeun helped to develop. All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by Teaching Assistantships from the Department of Computer Science and Engineering, Texas A&M University and research grant CCF-1423433 from National Science Foundation.

NOMENCLATURE

NoC	Network-on-Chip
VC	Virtual Channel
Flit	Flow Control Digit
CMP	Chip-Multiprocessors
RC	Remote Control Mechanism
NI	Network Interface
MC	Memory Controller
SoC	System on a Chip or System-on-Chip
HMC	Hybrid-Memory Cube
HBM	High Bandwidth Memory
NVM	Non-volatile Memory
NMP	Near-Memory Processing
PIM	Processing-in-Memory
DNN	Deep Neural Network
RL	Reinforcement Learning
AIMM	Artificially Intelligent Memory Management Unit Mechanism
MC^2sim	Memory Centric Computing simulator

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Data processing applications are evolving rapidly to process tera-bytes of data everyday, as it drives the modern consumer market. They are adopting kernels to not only handle huge amount of data to be processed in realtime, but also with utmost precision. To facilitate fast and accurate processing of data, depending on the type of applications, use of specialized hardware, capable of providing computation bandwidth are becoming common practice these days. On the other hand, computation bandwidth can only be utilized when data can be supplied to the processing units on time. Historically memory access bandwidth is inadequate for providing high processing performance, which could earlier be mitigated by the use of cache memory that exploits spatial and temporal data access locality. Contemporary applications like graph processing, machine learning applications, often exhibit sparse data access behavior, making caches ineffective. To exploit the full benefit of the processing capability and large internal memory bandwidth, memory-centric systems with network of memory chips are becoming more popular these days. For extracting performance by accelerating every aspect of the generalized processor is becoming prevalent, leading to realize the system by connecting many smaller specialized chips (chiplet) together to work in a harmony. This dissertation investigates ways to make this chiplet integration easy and free of communication hazards, like deadlock. It also investigates the data communication aspect of memory-centric systems followed by its data and computation mapping. Finally, we describe simulation framework that is designed to conduct the experimentation and also to contribute to the research community, as there are only a handful of memory-centric free source system simulator available right now.

1.1 Deadlock in Modular SoCs

With the advancements in silicon technology, Systems-on-Chip (SoCs) are becoming more complex and expensive, which motivates the designers to break the whole SoC into multiple small independent *chiplets* for reducing design cost and achieve a better scalability. The modular de-

sign of SoCs using 2.5D integration technology is a total paradigm shift from the monolithic SoC design to the hierarchical SoC design [3, 4, 5]. It allows to design smaller independent chiplets such as CPU, GPU, and accelerators with low cost and complexity, and integrate them together on an interposer, creating heterogeneous chiplet-based architectures. The chiplet-based design also increases the usability of chiplets in different SoCs and provides flexibility for vendors to manufacture using any desired process technology. We use modular SoCs and chiplet-based systems interchangeably.

One of the major concerns in any network-based system is deadlock due to cyclic hold-and-wait among virtual channels (VCs) [6]. Since chiplets are designed independently, their integration on an interposer brings new challenges to provide correctness validation. Connecting several deadlock-free NoCs together in a modular SoC may introduce a new kind of deadlock formed among different chiplets, as they are oblivious to each other's routing algorithm [7]. There have been many studies that address deadlock issues in conventional interconnection networks [6, 8, 9, 10, 11]. Conventional deadlock avoidance techniques cannot be applied directly to modular SoCs, as they consider the whole SoC as a single network, which violates the fundamental modularity principle of the chiplet-based system design.

Keeping the design modularity in mind, recently Yin et al. [7] propose Modular Turn Restriction (MTR), which imposes extra turn restrictions on the boundary routers of chiplets to avoid deadlocks in modular SoCs. MTR is easy to implement in hardware but needs changes in both chiplet routing and SoC routing. In addition, because of the skewed turn restrictions, this approach can lead to load imbalance and create several hotspots, which are detrimental for network throughput. From this work we notice that the boundary routers are playing a major role in inter-chiplet deadlocks.

We exploit two key insights regarding deadlocks in modular SoCs for providing a solution that works with any chiplet routing. First, outbound packets (going out of the chiplet) may block inbound (going inside a chiplet from outside) and intra-chiplet (source and destination in the same chiplet) packets to reach destinations. The other key observation is that packets involved in a

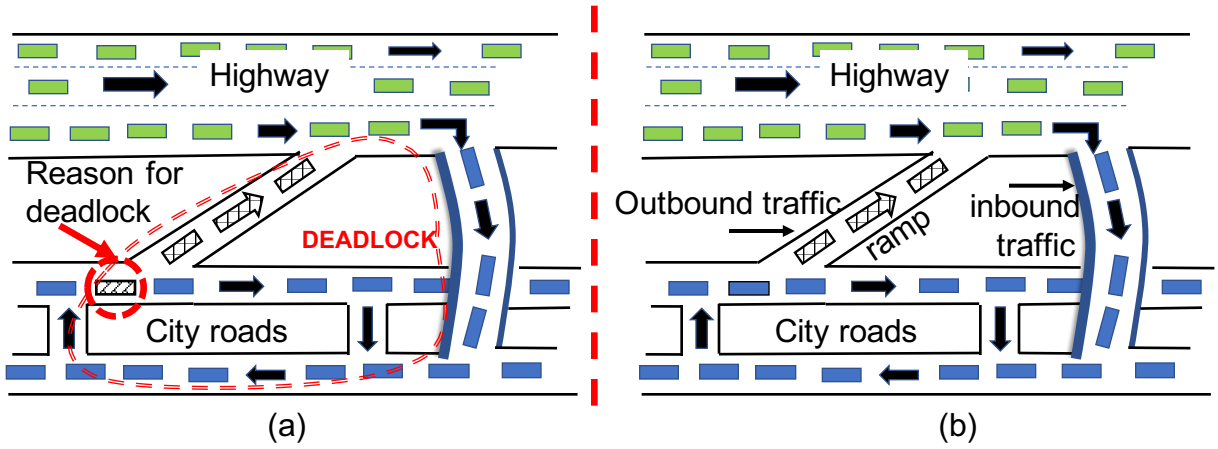


Figure 1.1: Analogy with a Road Situation. (a) Traffic is forming deadlock involving both highway and city-road traffic because of mutual blocking. (b) Once we make sure that all the highway-bound cars can stay in the ramp, until they get a space in highway, deadlock freedom is guaranteed. Note that in this case (also in out inter-chiplet deadlock) traffic isolation is enough to avoid deadlock, as highway and city-roads are deadlock free.

deadlock cross the boundary of the chiplets through a set of specific boundary routers[1]. Since the chiplets and interposer have independent deadlock free routing techniques, a deadlock is not possible in any of them separately, meaning it must involve both. Based on MTR and our observations, we pin-point the reason for this newly evolved deadlock among the chiplets in the modular SoC design. So we propose *Remote Control (RC)*, which is specific and highly optimized to solve this deadlock issue with minimum cost. It is also generalized enough to be applicable to chiplets with any kind of network, and SoC with any kind of chiplets connected in any topology with corresponding routing.

In Figure 1.1, we draw a real-life analogy with a road situation, where the highway is considered as part of the interposer network and city roads are considered as part of the chiplet network. Cars moving from city to highway are considered as outbound cars. The ramp in between the highway and the city roads is analogous to the existing output buffer (we named it as *rc_buffer*) in the boundary router. In Figure 1.1(a) it is clear that if all the outbound cars cannot be accommodated in the ramp, it may result in a deadlock. Therefore, we must get reservation for ramp (*rc_buffer*)

[1]Boundary Router: Chiplet routers that are connected with interposer router.

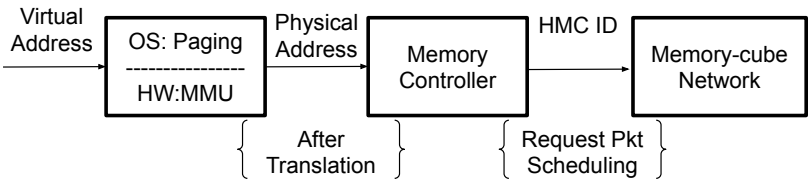
before getting down to city roads as shown in Figure 1.1(b) to avoid a deadlock. That means at any point there will be only those many outbound cars on the city road, which can be accommodated in the ramp between the city road and highway. We assume that the highway and city roads do not fall in deadlock if they operate in isolation. Hence we propose *RC* to control the injection of outbound packets to ensure isolation of these two types of traffic.

1.2 Data and Computation (Re)Mapping

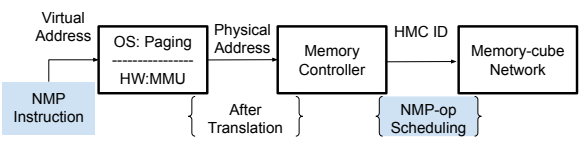
With the explosion of data, emerging applications such as machine learning and graph processing [12, 13] have driven copious data movement across the modern memory hierarchy. Due to the limited bandwidth of traditional double data rate (DDR) memory interfaces, memory wall becomes a major bottleneck for system performance [14, 15]. Consequently, 3D-stacked memory cubes such as the hybrid memory cube (HMC) [16] and high bandwidth memory (HBM) [17] were invented to provide high bandwidth that suffices the tremendous bandwidth requirement of big data applications. Although high-bandwidth stacked memory has reduced the bandwidth pressure, modern processor-centric computation fails to process large data sets efficiently due to the expensive data movement of low-reuse data. To avoid the high cost of data movement, near-memory processing (NMP) was revived to enable memory-centric computing by moving the computation close to the data in the memory [18, 19, 20, 21, 22].

Recently, memory-centric NMP systems demand even larger memory capacity to accommodate the increasing sizes of data sets and workloads. For example, the recent GPT-3 transformer model [23] has 175 billion parameters and requires at least 350 GB memory to load a model for inference and even more memory for training. As a solution, multiple memory cubes can be combined to satisfy the high demands [22]. For instance, the recently announced AMD Radeon VII and NVIDIA A100 GPU have 4 and 6 HBMs, respectively. More memory cubes are adopted in recent works to form a memory-cube network [24, 25] for further scaling up. Moreover, NMP support for memory-cube network has also been investigated in recent proposals to accelerate data-intensive applications [21].

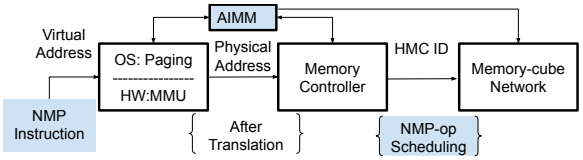
Along with the NMP and memory system developments, memory mapping for placing data



(a) Data mapping in conventional system.



(b) Memory (data/compute) mapping in NMP system.



(c) Memory mapping in NMP system with AIMM.

Figure 1.2: Memory mapping in different systems: (a) data mapping in conventional system, (b) data and compute mapping in NMP system, and (c) data and compute mapping in NMP system with AIMM.

and computation in the memory-cube network has become an important design consideration for NMP system performance. The paging system maps a virtual page to a physical page frame, then the memory controller hashes the physical address to DRAM address, which identifies the location in the DRAM. For NMP systems, the memory mapping should handle computation in addition to data, as shown in Figure 1.2b. Besides the data mapping, the memory controller also decides the memory cube in which the NMP operation is to be scheduled.

Prior work has focused on physical-to-DRAM address mapping to improve the memory-level parallelism [26, 27, 28]. Recently, DRAM address mapping has been investigated in NMP systems to better co-locate data in the same cubes [22]. However, adapting the mapping for the dynamic NMP application behavior is problematic due to the possibility of excessive data migration for every memory byte in order to reflect the new mappings. On the other hand, virtual-to-physical page frame mapping provides an alternative approach to adjust the data mapping during run time. Although such research has existed for processor-centric NUMA systems [29, 30, 31, 32], it has not yet been explored for memory-centric NMP systems, where computation is finer grained and tightly coupled with data in the memory system.

Furthermore, recent NMP research [18, 21] tightly couples the computation mapping with

given data mapping for static offloading, which has not considered the co-exploration of the intermingling between them. This may cause computation resource under utilization in different cubes and lead to performance degradation, especially for irregular data such as graphs [33] and sparse matrices [34]. Thus, it is challenging to achieve an optimal memory mapping for NMP with another dimension of computation mapping. Moreover, the unique and dynamic application behaviors as well as the intractable decision space with NMP in the memory-cube network make it even more challenging to design a universal optimal mapping for all types of workloads[2].

We propose an Artificially Intelligent Memory Mapping (AIMM), that optimizes data placement and resource utilization through page and computation remapping. AIMM can adapt to different applications and runtime behaviors by continuously evaluating and learning the impact of mapping decisions on system performance. It uses a neural network to learn the near-optimal mapping at any instance during execution, and is trained using RL, which is known to excel at exploring vast design space. In the proposed system, AIMM interacts with the paging system, the memory controller, and the memory-cube network. It continuously evaluates the NMP performance through the memory controller and makes data remapping decisions through the paging and page migration systems in the memory network. It is also consulted for computation remapping to improve NMP operation scheduling and performance.

1.3 Framework Development

Data processing applications' demand for high memory capacity grows with the increased data size, incurring disproportionate increase in the cost of data movement between the processor and the main memory in processor-centric computation systems. Majority of those computations are simple matrix multiplications that consist of dot-products followed by reduction or accumulation, which can easily be carried out in SIMD systems like GPGPUs. However, GPGPUs also incur the data movement between the main memory and GPU memory after the kernel is offloaded. This ever increasing demand of data movement gives rise of in-memory/near-memory computa-

[2]We conservatively calculate the decision space by taking the minimum number of pages touched (n pages) in any epoch in our set of applications, and mapping them to any of the m cubes, which constitutes = m^n, where $MIN(n) = 12$ and $m = 16$ in our system $\approx 10^{14}$ with the average value of n in an epoch ranges from $\approx 10s - 1000s$.

tion paradigm, which completely shunts the data movement between the processors and the main memory, as it supports computation inside the augmented memory module or near to the data location. The operation offloading is still needed from the processor side to the NMP memory side. In addition, it is not practical to convert all the memory accesses in an application into NMP operations, and hence, the systems still need to support conventional way of memory accessing.

In terms of memory technologies, the memory-centric design can be categorized into (1) DRAM based, (2) NVM based, and (3) SRAM based [35]. The DRAM based designs are again divided into (1) commercial DIMMs [36, 37, 38] and (2) 3D DRAM cubes. The 3D DRAM has two distinct approaches, (1) HBM [39] and (2) HMC [18, 19, 21, 40, 41, 42]. The NVM based designs use ReRam [43, 44]. Even though there are a lot of variations in the memory technology, they also share common traits for building the memory system. For instance, for ease of capacity scaling they tend to have modularized design, where certain size of modules are connected through interconnection network. However, it is evident that the functionalities of modules of different technologies widely vary, and so as their control and management systems.

There are many hard problems in the architecture for which historically we had either heuristic based approximate runtime solutions, or complex and customized offline solutions for deciding a rigid hardware policy. For instance, finding the optimal cache replacement policy, designing optimal prefetcher, or optimal thread scheduler, etc., considered as hard problems to solve. Recently by leveraging advanced technology artificially intelligent schemes [45, 46, 47, 48, 49, 50, 51] have been proposed to go hand-in-hand with conventional hardware systems. So hardware design need to adapt to the required changes to facilitate the machine learning framework integrated in it.

Simulators have been prevalent tools in the computer architecture research community to validate innovative ideas, as prototyping requires significant investments in both time and money. Many simulators have been developed to solve different research challenges, serving their own purposes. Many memory-centric processing researchers either using in-house simulators customized to their requirements, or using some modified versions of cpu-centric computation simulators available for several years. Developing simulation framework needs both time and effort and hence we

immediately need a framework generalized enough to be used to evaluate different innovative ideas on different technologies and easily extendable to implement new policies. In addition, if we have an evaluation standard, it will be easier to compare across the techniques quantitatively even without spending time to reproduce the results, which should lead to overall higher research throughput.

1.4 Thesis Statement

For exploiting the complete benefits of NMP using modular SoC, the integration must be guaranteed to be deadlock free for providing better design flexibility to both chiplet and SoC designers and NMP data/computations should be intelligently placed in the memory to minimize the NMP operation costs.

1.5 Contributions

- *RC* guarantees deadlock freedom in modular SoC by simply controlling a subset of packet's injection from a remote node in the same chiplet. Any chiplet topology and routing can adapt *RC*, making it flexible and attractive for practical purposes. We formally prove that *RC* guarantees deadlock freedom in modular SoC using an illustrative generalized example. We quantitatively show that *RC* is better than the state-of-the-art technique(s) and to the best of our knowledge, this is the first work that uses injection control based technique to guarantee deadlock freedom in the modular SoC design domain.

- AIMM, an artificially intelligent memory mapping, continuously learns and adjusts the memory mapping dynamically to adapt to application behavior and to improve resource utilization in NMP systems. To the best of our knowledge, this is the first NMP work that targets mapping of both data and computation. In AIMM a reinforcement learning formulation explores the vast design space and used to train a neural network agent for solving the memory mapping problem. We show a detailed hardware design and practical implementation of RL directed data and computation mapping in a plug-and-play manner to be applied in various NMP systems. A comprehensive set of experimental results show that the

proposed AIMM improves the performance of state-of-the-art NMP systems significantly.

- We introduce MC^{2}sim as a memory-centric computation framework, with plug-and-play HMC- and HBM- network through a set of memory interfaces, and easily extendable to NVM technologies. We also plug-in a reinforcement learning (RL) agent (can be configured to disable if not required), which is capable of reading architectural data and can be interfaced with any hardware policy making through defined interfaces. The simulator is highly optimized for high simulation speed to make RL training and inference possible in parallel to the hardware simulation. Most of the simulator components are highly parameterizable.

1.6 Dissertation Organization

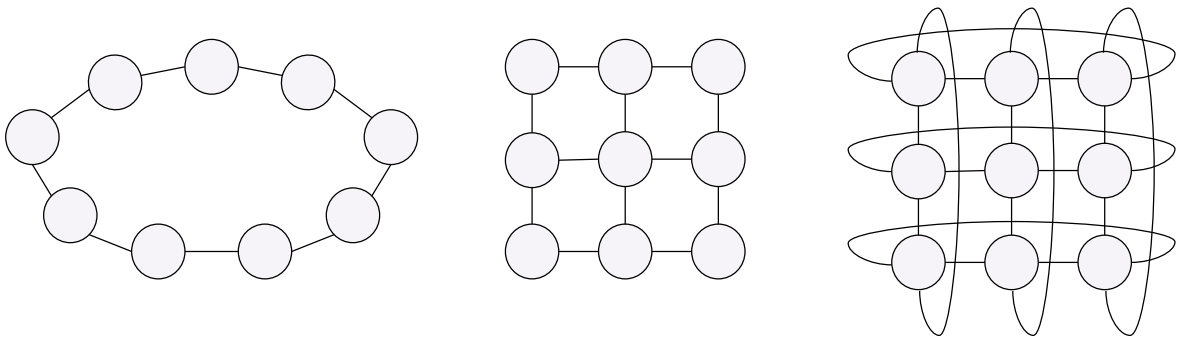
The rest of the dissertation is organized as follows. Chapter 2 introduces the fundamental concepts of interconnection network, deadlock, SoC, SoC deadlock, different aspects of the processing-in-memory and memory-centric computing framework design, to lay down a foundation for the discussion of this research, and survey the literature of related work. Chapter 3 presents the SoC deadlock issues and proposed solution, Chapters 4 and 5 present generalized data and computation mapping solution for the NMP techniques and framework design for the evaluation of memory-centric designs with NMP. Finally, Chapter 6 concludes this dissertation with a discussion on the future research directions.

2. BACKGROUND AND LITERATURE SURVEY

In this chapter we present the fundamental concepts of interconnection network and deadlock problem associated with it, followed by SoC and deadlock observed in modular SoCs. Then we discuss about the 3D stacked DRAM fundamentals, along with data and computation mapping aspects of in-memory/near-memory processing, followed by a brief discussion on the existing simulation frameworks and motivation for design a new memory-centric computing framework.

2.1 Interconnection Network Basics

Interconnection network being the backbone of any multi-node system, plays a major role in system performance. The communication fabric is built by connecting several routers through physical link channels. Network topology, routing algorithm, flow control protocol and router microarchitecture implementation are the major aspects of any network interconnect design. Throughout this dissertation we design and evaluate several types of networked systems, namely CMP network, SoC chiplet and interposer network, memory cube network, etc. We discuss each basic aspects of the network design, which is applicable for any kind of network, briefly as follows.



(a) Ring topology (b) Mesh topology (c) Torus topology

Figure 2.1: Three topology examples: ring (a), mesh (b), and torus (c).

2.1.1 Topology

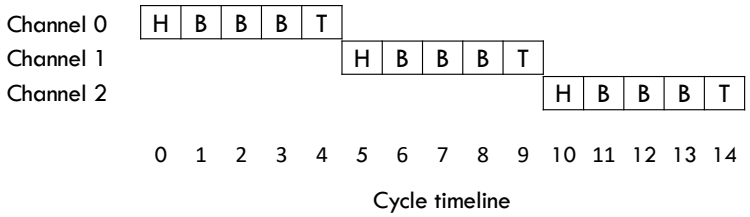
The network topology forms the shape of the network by defining its connections. Figure 2.1 shows different types of topologies as an example of symmetric topology. There can be asymmetric or irregular topologies as well. Router radix dictates the freedom of the topology as number of routers that can be connected with any router is constrained by the router radix. The ring topology is constructed by utilizing the low radix routers whereas the torus demand high radix routers to form the network, and depending on that the network performance and energy consumption are decided. The performance and energy trade-off sometimes lead to right choice for a network topology.

2.1.2 Routing Algorithm

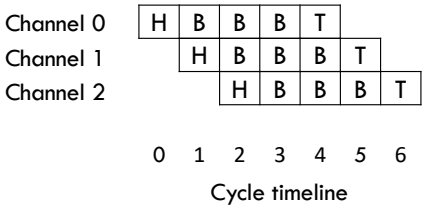
Routing algorithm goes hand-in-hand with the topology as each topology needs a customized routing algorithm, which must ensure that the traffic in the network never falls in the deadlock. Routing algorithm can be either deterministic or non-deterministic. Deterministic routing algorithms are popular to be simple to implement using minimum resources. On the other hand, the non deterministic routing algorithms are known for providing path diversity for routing packets and so as the network throughput. Routing algorithms are prone to deadlock as one routing packet tend to reserve multiple of network resources for achieving high performance, resulting in circular hold and wait often involving multiple packets in circular hold-and-wait. The deadlock situation can be solved mostly in two ways, (1) deadlock avoidance [8], and (2) deadlock recovery [52, 9]. We detail the deadlock problem and solutions in following sections.

2.1.3 Flow Control

Flow control is analogous to traffic lights on the roads. Similar to the traffic lights controlling cars' movements, flow control manages the flow of the packets in the network, so that all the packets can reach to their corresponding destinations safely. Generally The flow control technique regulates the flow of the traffic between two routers. However, there could be indirect ripple effect of flow in different pair of routers on each other, mostly as the backpressure in the opposite



(a) Store-and-forward example



(b) Virtual cut-through example

Figure 2.2: Flow control examples at low load without contention: store-and-forward flow control (a) and virtual cut-through flow control (b). Figures are redrawn and adapted from examples in [1], where `H` indicates head flit, `B` and `T` stand for body and tail flits, respectively.

direction of the traffic flow under consideration. It is evident that stricter flow control may have a negative impact on the network performance resulted due to network stalls. On the other hand, relaxing the flow control may make the network deadlock prone.

In the Figure 2.2 two examples of flow control techniques are illustrated. (a) store-and-froward stores the whole packet and then only forwards the packet to the next node in the network (downstream router). The technique is simple yet detrimental for the network performance as the packets are practically sent as the form of several flits through the physical channels. The width of the physical channels are in general equal to the flit width, and hence transferred through the channel from upstream router to the downstream router within one network cycle. (b) virtual cut-through can be considered as improvement over the store-and-forward technique, where the whole packet need not reach to the VC buffer for forwarding already reached flits further downstream. Only downside of this flow control is that the buffer sizes are bigger or at least equal to the packet size. This shortcoming is addressed in the wormhole flow control which works with any size of the buffer as the credit-flow happens in the flit level as opposed to packet level in virtual-cut-through and hence likely to provide a high network throughput.

2.1.4 Router Microarchitecture

The router is responsible for routing packets so that the packets reach to their destinations through the network. Router micro-architecture (shown in Figure 2.3) facilitates the packet receive, evaluating its destination to route them forward. In order to do that each router try to reserve buffer

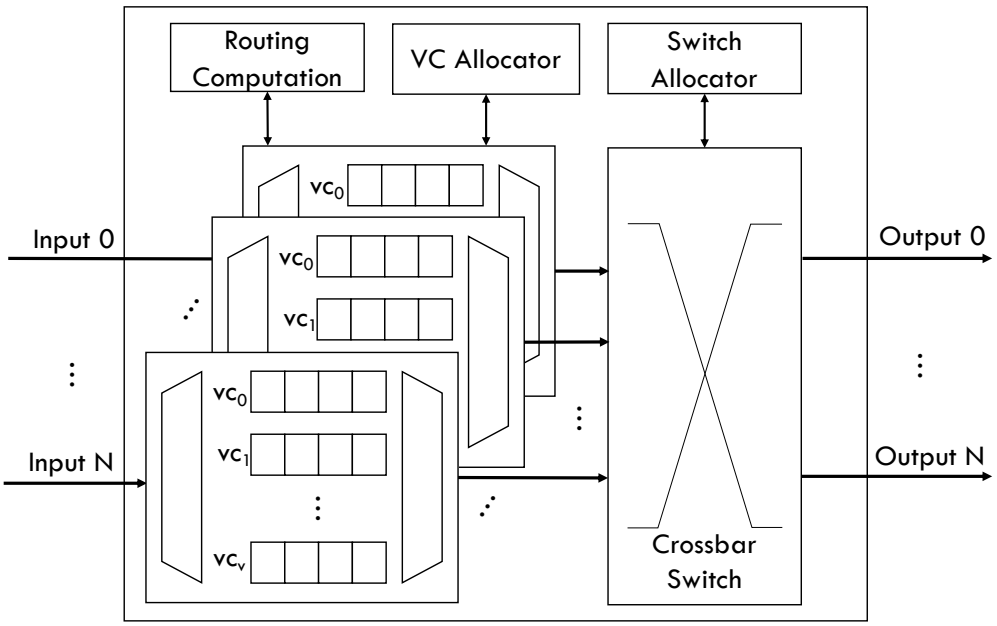


Figure 2.3: Virtual-channel router microarchitecture.

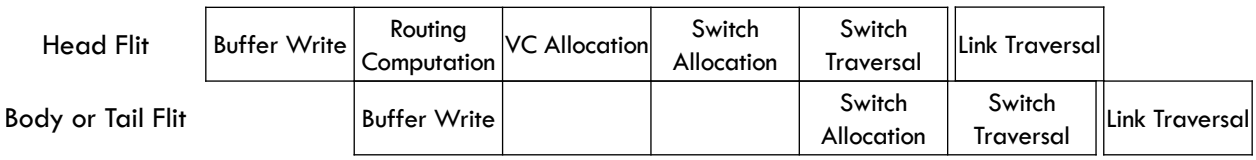


Figure 2.4: Traditional 5-stage router pipeline followed by link traversal.

in the downstream router by invoking VC allocation. Since one router may be connected with multiple other routers, often they need to arbitrate the packets when they try to go to the same destination in the same cycle. Switching/Switch-traversal is considered as the integral part of the packet sending process which is followed by the switch allocation and arbitration. Link traversal is followed by the switch traversal. In Figure 3.3 we show a conventional five stage pipeline router functionalities step by step.

2.1.5 Network-on-Chip (NoC) Deadlock

Network-on-Chip (NoCs) are on-chip interconnection networks in silicon that interconnect execution cores, slices of cache, memory and IO controllers. Deadlock being a pressing network

issue for long time, it is well known and well exercised. It occurs mostly because of circular hold and wait for the network resources across the network. Here we briefly discuss the state-of-the-art deadlock avoidance mechanisms in the context of monolithic NoC and later extend our discussion for deadlock in hierarchical network in modular SoCs. Deadlock avoidance mechanisms fan out in two distinct branches, namely VC and turn model based, and flow control based techniques. The first type either rely on turn restrictions, or on dedicated/ordered VC buffer for different traffic types/directions. On the other hand, flow control techniques either control the injection of packets, or ensures bubble in the buffer to avoid deadlocks.

2.1.5.1 Turn Restriction and VC over-Provision

Duato proposed escape-VC [8], a theory for deadlock freedom for routing with cyclic channel dependency. Duato's theory can be applied for both deadlock avoidance [53, 54] and deadlock prevention [55, 52, 56] techniques. Idea of escape channel cannot be applied directly in modular SoC as the packets in the escape-VC must be propagated using a deterministic deadlock free algorithm, which cannot be guaranteed in a modular SoC. Recently Ebrahimi et al. [57] propose *EbDa* that provides exclusive sets of VCs to isolate traffics (say, intra-chiplet traffic, and inter-chiplet, or outbound traffic) to avoid deadlocks. However, VC separation leads to lower utilization and is shown less attractive in MTR [7]. Dally et al. [6] propose to use two or more VCs in order to avoid the cyclic channel dependencies. It ensures deadlock freedom by using total ordering of VCs. Even though this condition is sufficient to avoid the deadlock, it is not necessary [9]. Extra VCs result in increase in the router area and energy consumption. Based on Dally's theory, a few other techniques have been proposed that use additional VCs [58, 59, 60] to avoid deadlock. Another way to achieve strict order of reservation for the shared VCs is by imposing turn restrictions [61, 62, 63] on the packet traversal.

2.1.5.2 Flow Control Based

For providing deadlock freedom, flow control techniques either regulate the injection [64, 65] of the packets or allow a packet to go forward depending on the buffer occupancy [66] in the ring.

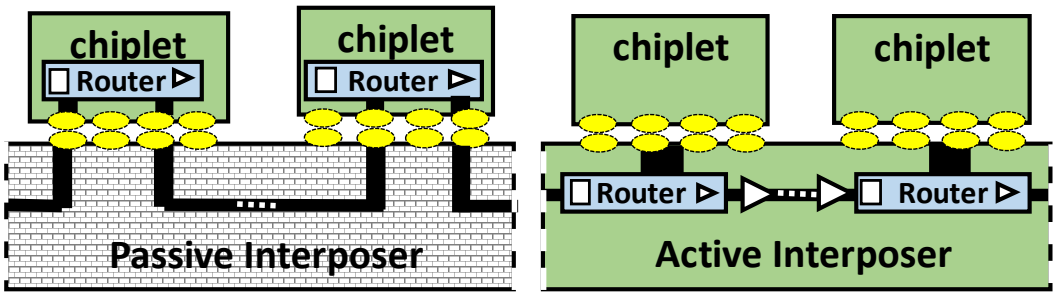


Figure 2.5: Connecting multiple chiplets using passive and active interposer [2]. No logic can be implemented in passive interposer. Unlike passive interposer, we can implement any logic in active interposer. For instance routers are implemented in active interposer for connecting chiplets.

The second concept is coined as bubble flow control by Puente et al. [67] and applied in torus network for the flow control in escape channel. This concept is being used in in-transit buffer for avoiding deadlock in k-ary n-cube torus network [68], and extended later for irregular off-chip network [69, 11], worm-whole switching [70], torus cache-coherent NoCs [71]. We take the wisdom from bubble flow control and use that in *Parking* lot design. We also use the concept of injection control for outbound packets to bound the *Parking* lot size (even to a single packet buffer). Recently Ramrakhani et al. [9] propose *SPIN* to recover from deadlock by making packets in the deadlock loop, move hop by hop in a synchronous progress. It is very challenging to apply *SPIN* in modular SoC, where the chiplets are designed independently, and connected through the interposer routers. Moreover, synchronization of packet movement among chiplets make the design very complicated.

2.1.6 Modular 2.5D SoC Integration

Modularity has been advocated as a new design principle to reduce the complexity and cost of the SoC design. An SoC is called modular if all the chiplets on that SoC are designed independently. Contemporary multi-chiplet SoC integration uses a passive silicon interposer [72], where the only way to make connections between chiplets is to make fixed wire connections as shown in Figure 2.5. In a passive interposer, dedicated wire connections are required from a chiplet to connect with different chiplets [2]. This may lead to long wire usage with multiple repeaters and

a huge number of dedicated communication channels, making it hard to scale in terms of area and energy. In addition, the channels in a passive interposer should be standardized for the modular SoC design. Hence, there has been an increase in research of active interposers [73, 4, 74] both in industry and academia. We also consider an active interposer substrate for designing the interposer network. Active interposer facilitates interconnection between the chiplets [2, 75, 76] by adopting the router design in the silicon substrate, which is more area-and energy-efficient. The integration process is generally known as 2.5D integration, featuring a silicon interposer as shown in Figure 2.5. It is placed between the System-in-Package (SiP) substrate and the dice, where this silicon interposer has Through-Silicon-Vias (TSVs) connecting the metalization layers on its upper and lower surfaces [76].

2.1.7 Deadlock Freedom in Modular SoC

2.1.7.1 Modular Turn Restriction (MTR)

Based on the principle of turn restrictions [61], recently, Yin et al. [7] propose a deadlock-free routing algorithm for modular SoC. As the best of our knowledge, this is the only work on modular SoC deadlock freedom so far. At design time, MTR finds the optimal placement and turn restrictions for boundary routers of each chiplet independently with the help of Channel Dependency Graph (CDG) analysis. The turn restrictions are applicable to both the packets that go out from the chiplet (*outbound packets*) as well as to the packets that reach from other chiplets (*inbound packets*). Once the list of turn restrictions is obtained, MTR applies the turn restrictions in the chiplet routing, which are applicable only for the outbound packets. For imposing turn restriction on the inbound packets, interposer routing also needs to be modified, which imposes constraint on the SoC designers and increases design complexity.

2.1.7.2 VC Separation (VC-SEP)

The idea of VC separation is widely applied to avoid protocol deadlock as well as routing deadlock based on Duato's theory [56]. We showcase it as a potential solution for SoC deadlock since it is a natural fit for this particular problem. The traffic in the Modular SoC can be categorized into

two. (1) *Inter-chiplet*: traffic of packets which do not have destinations in the source chiplet. (2) *Intra-chiplet*: traffic of packets having both sources and destinations in the same chiplet. Without any constraint on the area, cost and energy, VC-SEP naturally segregates two different traffic by providing two different virtual networks throughout the system. For outbound packets, we allocate first half of the total set of VCs, and other set of VCs are being allocated for inbound packets and all the intra-chiplet packets.

2.1.7.3 In-Transit Buffer (ITB)

The idea of ITB is originally used by Flich et al. to avoid deadlocks in irregular networks and later extended for off-chip networks in the cluster of workstations [11]. Here, we adopt the idea of ITB and apply in the Modular SoC to avoid deadlocks. ITB uses the Network Interface Card (NIC) memory as an in-transit buffer in some pre-decided nodes. Those special nodes are being selected after CDG analysis as deadlock breaking points. Any packet that reaches to those nodes are forced to eject in that node. Using DMA, the whole packet is stored in the NIC memory. In case the NIC memory gets exhausted, the packet is dropped and a NACK packet is being generated and sent to the source node for re-transmission. This process continues till the packet is ejected successfully in that special node. If the packet is successfully stored in NIC memory, the NIC sends an ACK message to the source node. To port this idea in the Modular SoC, we consider the boundary routers as special nodes and use the Network Interface (NI) connected to boundary routers to place ITB, a small buffer to store packets (no DMA) in the similar way described above. The ejection and reinjection in some special nodes (boundary routers) break the circular channel dependency chain and hence avoid deadlocks. By using domain specific knowledge we further optimize the performance of this implementation by doing ejection-injection only when a packet is outbound. Please note that we made necessary changes to the original implementation for accommodating the idea of ITB into the modular SoC context.

2.2 3D stacked DRAM and NMP

2.2.1 Die-Stacked Memory.

HMC [16] and HBM [17] with logic and DRAM dies in the same silicon have enabled high bandwidth memory-centric design and near-memory processing. HMC uses packet-switching protocol to simplify capacity scaling by chaining multiple cubes as a memory-cube network [21]. On the other hand, HBM is tightly coupled with the processor using direct connections and provides the flexibility of memory controller design for command scheduling. In this work, we evaluate AIMM on a system with large chained HMC network, and also conduct a case study in §4.5 on capacity scaling using recently proposed HBM-PIM [39] in context of NMP operation and data (re)mapping.

2.2.2 Memory Network

Conventional systems with DDR memory have capacity limits and bandwidth bottlenecks due to the limited number of pins per processor chip. Therefore, it requires more processor sockets in such systems to scale their memory capacity. However, the overweight data movement with respect to light computation in emerging data-centric applications can lead CPU to be under-utilized. In contrast, HMCs can be chained together to form a cost-effective memory network using packet switching and provide large memory capacity. In addition, commonly adopted processor-centric design optimizes processor-to-processor communication but overlooks the overall system bandwidth utilization. A recent study [24] has shown that memory-centric designs can achieve better bandwidth utilization as compared to processor-centric designs.

2.2.3 Near-Data Processing.

Recently, a significant amount of research efforts to reduce data movement across the memory hierarchy to improve the system efficiency. Near-data processing (NDP), as a promising compute paradigm, has driven new architectures to move computations near data-resident locations, such as cache and memory. Aga et al. proposed compute cache [77] that uses bit-line circuit technology to perform simple computation in the cache to enable in-place computing. Processing-in-memory

(PIM) [19, 78, 79, 20, 80, 81, 82, 83, 22] is an alternative NDP design that introduces compute elements in memory for data processing. Recent studies [18, 84] have proposed to integrate PIM architectures within modern systems in a seamless fashion. They extended the instruction set to offload computations to data-resident memory modules. These mechanisms achieve better efficiency compared to conventional computing due to reduced data movements. They are most effective in the case of irregular memory accesses and atomic write operations. However, they are suboptimal when performing simple tasks over a large size of raw data, such as *dot product*, since they need to fetch part of the data across the memory network for further processing when data are not located in the same module that incurs communication and energy overhead. Ahn et al. proposed Tesseract [19], a programmable PIM accelerator for large-scale graph processing. Nair et al. [85, 78] proposed Active Memory Cube (AMC) by leveraging HMC to place vector processing units in the logic layer. AMC suffers from delays due to instruction pre-loading as well as delay and energy overhead of its complex interconnection network. Most recently Fujiki et al. [86] propose a programmable in-memory processor architecture, and data-parallel programming framework using non-volatile memory. Mondrian [82] takes an algorithm-hardware co-design approach to sequence irregular accesses for better locality. Recent study [87] analyzed Google workloads and discovered the data movement as the bottleneck for performance and energy efficiency, which is also the problem this research tries to solve.

The memory-centric design paradigm can be broadly classified as (1) processing using memory (PUM) [42, 88, 89, 37] and (2) near-memory processing (NMP) [90, 91]. PUM typically augments memory circuit for analog computation in the memory [42, 88, 89, 37] or cache [35]. We focus on NMP in this work, where processing units (PUs) are built on the logic layer of HMC [18, 19, 21], separate from the DRAM layers. In NMP, operations are offloaded from the processor to the memory in various granularity, ranging from operation-level [20, 18], to cache block [21], page [92], and kernel level [19, 22], where NMP system complexity varies accordingly. As operation-level offloading is the most flexible and practically feasible [39] in an augmented memory system, we choose those NMP systems as our primary target for data and computation mapping optimization.

Prior research [93, 94, 95] has advocated to provide computation power as well as routing functionalities in communication fabrics. Active Message [93] embeds the function pointer and arguments across the network to perform tasks in remote compute nodes. Pfister et al. [94] and Ma [96] proposed mechanisms to combine messages so as to reduce network traffic. Recently, IncBricks [95] implements an in-network caching middlebox for key-value acceleration in router switches. Several studies [97, 98, 99] proposed mechanisms to optimize shared value update or reduction in the network. The NYU Ultracomputer [97] introduced adders in routers to combine fetch-and-update requests for the same shared variable. Panda [98] and Chen et al. [99] proposed similar hardware to optimize reduction in the network interface for MPI collective communications. These mechanisms only support pure reduction operations and cannot accelerate operations like *dot product*, thus requiring significant data movements across the memory hierarchy to first compute the intermediate results. Recently, Kwon et al. proposed MAERI [100] to improve efficiency for data-flow computations in deep neural network accelerators, which does not target general applications. The multiply operations require data to be brought to local SRAM and are calculated only at leaf nodes in the tree-based network topology. These in-network compute solutions have limited adaptivity since the reduction tree/ring is statically tied to the network topology.

2.2.4 Computation and Data Mapping in NMP

Existing NMP proposals such as PEI [18] and Active-Routing (AR) [21] focus more on offloading and hardware design while leaving computation and data mapping unexplored. For computation mapping, the location decisions and operation offloading schemes vary across different techniques. For instance, AR sends the operations as `<&dest += &src1 OP &src2>`, where each of the operands carry their physical addresses. On the other hand, PEI fetches one of the source data from the cache and sends the request along with that data. The computation point for each of the operations is decided following fixed policies, which either maps to closest to both the sources, or at the destination. The result of the NMP operation is returned to the processor as a response. We evaluate AIMM on two variations (created based on their computation points) of AR and PEI described in §5.5.

For data mapping, AR and PEI rely completely on the OS for data allocation and are oblivious to the data location. They can reduce data movement between the processor and the main memory significantly, but do not address the cost of individual operation with operands being spread across the memory network. To address this issue, we employ modified TOM [22], as a heuristic-based solution, which originally facilitates data mapping for GPU NMP systems. TOM's mapping decision is applied to a host of pages and it fails to adapt to fast changing application behaviors such as graph processing. Owing to the observation that NMP applications exhibit a fine-grained relationship between the data and computation location, more adaptive and robust solution is required.

2.2.5 Data Locality in NUMA and Multi-Program Workloads

System supports for data and computation mapping for NMP is still in its infancy. In NUMA systems, heuristic-based approaches were proposed to co-locate data and computation for better locality, including different aspects of data allocation optimization and reallocation using migration [101, 102, 103, 104, 105, 106, 107, 108, 109, 110]. Multi-program workloads make the problem even more complex due to inter-process interference. In traditional multi-program workload systems, a thread-private allocator like HOARD [101] can be used to improve the data locality by ensuring the proximity of memory accesses in a thread through bulk allocation and per-thread free lists. In this work, we explore AIMM on top of HOARD in multi-program scenarios, intending to reduce inter-process interference for NMP-ops in addition to data and computation location optimizations.

2.2.6 Migration and TLB shootdown

Apart from migration in CPU-based NUMA [110, 111], page migration is also proven to be useful in multi-GPU shared memory systems. The "first-touch" approach [112, 113, 114] can localize memory allocation to the first thread or the first GPU that accesses the memory, but does not guarantee the locality in the future computations. Baruah et al. [115] proposed to select pages and migrate them at runtime to the highly predicted GPU. In AIMM, we co-explore both data movement and computation scheduling to improve resource utilization.

Page migration needs to update the page table, typically accompanied by a TLB shootdown for maintaining TLB coherence [116]. Commercial x86-64 processors do not support automatic invalidation of stale TLB entries on a page table update. Instead, the OS is responsible for TLB coherence maintenance via the expensive TLB shootdown process, which is an active research area [117, 118, 119]. In this paper, we use hardware-assisted TLB shootdown by employing a lazy invalidation approach as described in §4.2.3.

2.2.7 Data/Computation (Re)Mapping using RL

Reinforcement Learning (RL) is a machine learning method where an agent explores actions in an environment to maximize the accumulative rewards [120]. In this paper, RL is used to design an agent that explores data and computation remapping decisions (actions) in the NMP system (environment) to maximize the performance (rewards). Figure 2.6 shows a typical setting of RL, where an agent interacts with the environment $\mathcal{E}$ over a number of discrete time steps. At each time step t, the agent observes an environment state $s_t \in \mathcal{S}$ *(state space)*, and selects an action $a_t \in \mathcal{A}$ *(action space)* to change the environment according to its policy π, which is a mapping from s_t to a_t. In return, it receives a reward r_t and a new state s_{t+1} and continues the same process. The accumulative rewards after time step t can be calculated as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{2.1}$$

where $\gamma \in (0, 1]$ is the discount factor and T is the time step when the process terminates. The state-action value function $Q_\pi(s, a)$ under policy π can be expressed as

$$Q_\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a] \tag{2.2}$$

the objective of the agent is to maximize the expected reward.

In this paper, we use Q-learning [121] that selects the optimal action a^* that obtains the highest state-action value (Q value) $Q^*_\pi(s, a^*) = \max E_\pi[R_t | s_t = s, a_t = a^*]$ following the *Bellman*

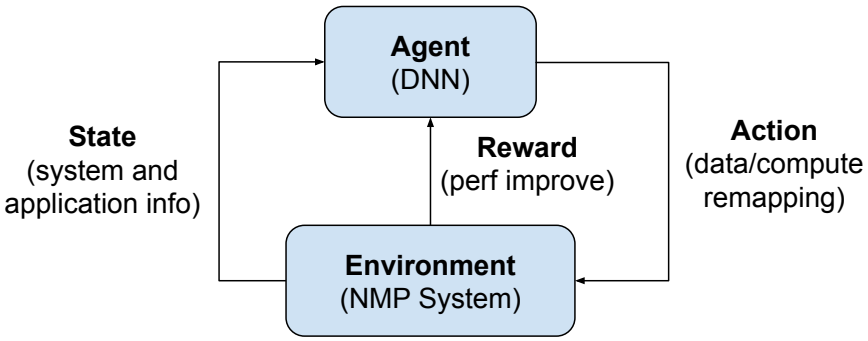


Figure 2.6: A typical reinforcement learning framework.

equation [120]. The Q value is typically estimated by a function approximator. In this work, we use a deep neural network with parameters θ as the function approximator to predict the optimal Q value $Q(s, a; \theta) \approx Q^*(s, a)$. Given the predicted value $Q(s_t, a_t; \theta)$, the neural network is trained by minimizing the squared loss:

$$L(\theta) = (y - Q(s_t, a_t; \theta))^2, \tag{2.3}$$

where $y = r_t(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a'; \theta | s_t, a_t)$ is the target that represents the highest accumulative rewards of the next state s_{t+1} if action a_t is chosen for state s_t.

RL has been widely applied to network-on-chip (NoC) for power management [122], loop placement for routerless NoC [123], arbitration policy [124], and reliability [125]. It has also been used for designing memory systems, such as prefetching [126] and memory controller [51]. Additionally, it has been applied to DNN compilation and mapping optimization [127, 128, 129]. In this work, we use RL for co-exploration of data and computation mapping in NMP systems.

2.3 Framework Design

2.3.1 Why Memory-Centric Computing Simulator?

There are a large number of simulators available, which are designed following processor centric design principle as shown in Table 2.1, either focusing some specific components in the system (like booksim [130], DRAMsim [131], GPGPUsim [132], CasHMC [133], etc.), or the system

Simulator	FS/A	DC	μAr	Mc	SS	MCD	ML	Simulator	FS/A	DC	μAr	Mc	SS	MCD	ML
gem5 [134]	FS	N	Y	Y	+	N	N/A	**SimpleScalar** [140]	A	N	Y	N	++	N	N/A
GEMS [141]	FS	Y	Y	Y	+	N	N/A	**Booksim** [130]	N/A	N/A	Y	N	++	N	N/A
MARSSx86 [142]	FS	Y	Y	Y	+	N	N/A	**Gernet** [143]	N/A	N/A	Y	N	++	N	N/A
SimFlex [144]	FS	Y	Y	Y	+	N	N/A	**GPGPUsim** [132]	A	Y	Y	N	++	N	N/A
PTLsim [145]	FS	Y	Y	Y	+	N	N/A	**DRAMsim** [131]	N/A	N/A	Y	N	++	N	N/A
Graphite [146]	A	Y	N	Y	+++	N	N/A	**Dinero IV** [147]	A	N	Y	N	++	N	N/A
SESC [148]	A	N	Y	N	++	N	N/A	**Zesto** [149]	A	N	Y	N	+	N	N/A
Sniper [150]	A	Y	N	Y	+++	N	N/A	**CMP$im** [151]	A	Y	Y	N	++	N	N/A
MCSimA+ [135]	A+	Y	Y	Y	++	N	N/A	**Zsim** [136]	A	Y	Y	Y	++	N	N/A
Multi2Sim [152]	A+	N	Y	Y	+	N	N/A	**CasHMC** [153]	N/A	N/A	Y	N/A	++	N	N/A
PIMSim [137]	FS/A+/A	N	Y	Y	+/++/+++	Y	N/A	**MultiPIM** [138]	A+	Y	Y	Y	++	Y	N/A
NAPEL [139]	A	Y	N	-	+++	Y	Y	**MC^{2}sim**	**A**	**Y**	**Y**	**Y**	**+++**	**Y**	**Y**

Table 2.1: Summary of existing simulators in terms of some of the important features. (FS/A)-Full-system (FS) VS. application-level (A), (DC)-Decoupled functional and performance simulation, (μAR)-Microarchitecture details, (X86)-X86 ISA support, (Mc)-Manycore support, (SS)-Simulation speed, (MCD)-memory-centric design, (ML)-in-built machine learning support, (Y)-yes, (N)-No, (P)-partial, (N/A)-not applicable.

as a whole keeping processors at the center of the design (like gem5 [134], MCsimA+ [135], Zsim [136], etc.). There is always been a trade off between the simulator accuracy and the simulation time. That is why in general practice the main focus of the design is done in details and other essential supporting components are designed in a much abstract way to save simulation time.

Very recently we came across some efforts for supporting NMP, namely PIMSim [137], MultiPIM [138], NAPEL [139], which shows growing demand for memory centric design simulation tool. In MC^{2}sim design we tradeoff accuracy for speed as we are designing huge SoC with large number of cores, connected to 100s of GBs of main memory supporting NMP and act as environment for the reinforcement learning agent if enabled. In MC^{2}sim our main focus is accurate simulation of the NMP operations in the main memory system. Hence, we abstract the Chip multiprocessor side carefully without hurting its high level functionality and throughput. As many essential components are implemented functionally, providing absolute end-to-end execution time is not our primary goal, rather we intend to quickly and fairly compare merits of different NMP techniques on this platform.

2.3.2 Memory Centric Design

Memory-centric design being in its infancy there are only a few general framework available to evaluate different NMP techniques. Researcher customize the available stand alone different component simulators and integrate them accordingly to make them work with their NMP techniques. Most of the time the development takes time and also may have observe some secondary impact from the original processor centric design, often goes undetected.

2.3.2.1 Inherent NMP support

NMP system being a unique problem itself involves co-design of the memory and the computations in-order to make them work in a harmony. If the NMP is implemented in a scale-out large memory system it also involves unique remote request generation and receive response functionalities engaging network components, which in turn influences the flow control. If the simulation framework inherently support the NMP-op simulation, implementing different NMP-op policies should be much easier and less error prone that implementing from scratch.

2.3.2.2 Huge Memory Allocation and Access APIs

Another goal of memory-centric design is to scale the memory capacity to terabytes and still able to simulate the memory access behavior for the applications. As the best of our knowledge the current simulators do not support programming kernels on their own, rather rely on reading for the compiled binaries. The main downside of that is the memory allocation is access is limited by the host system configurations. We change that concept by integrating the programming interface with the simulation interface to support any size of memory allocation and access behavior simulation through customized memory allocation and access APIs, which works along with usual C++ programming constructs.

2.3.3 Simulation Speed

Simulation speed always been a concern for any simulator, which is almost a necessity for MC^{2}sim because of mainly two reasons, (1) intend to simulate memory access behavior of huge

memory, (2) intend to support machine learning integration and providing feedback for learning. Integrating existing cycle accurate simulators and by tweaking their implementation would provide us accuracy at the cost of simulation speed, as indicated in the Table 2.1. Moreover, we require even better simulation speed than any individual simulators listed.

2.3.3.1 Full System versus Application Level

The high accuracy of the full system simulators allow us to even count on absolute execution time at the cost of their simulation time. On the other hand for fair comparison across NMP techniques we do not need to have cycle accurate processor as more than 90% of the simulation is going to happen on the memory system side. That is why MC^{2}sim is somewhere in between the full system and application level as it implements some part of the system hiding hardware details, where as other parts in the NMP critical path are implemented in more details.

2.3.3.2 Decoupled Functional and Performance

To keep the RL agent as a plug-and-play module and easily replaceable, we decoupled its functional simulation from the NMP-op performances simulation. Moreover, it also boosts the simulation time as opposed to timing simulation of RL training and inference. However, we employ a clever phase shift policy where RL and simulation always go in one phase difference to give a parallel simulation view between the RL and corresponding hardware simulation.

2.3.4 ML Support

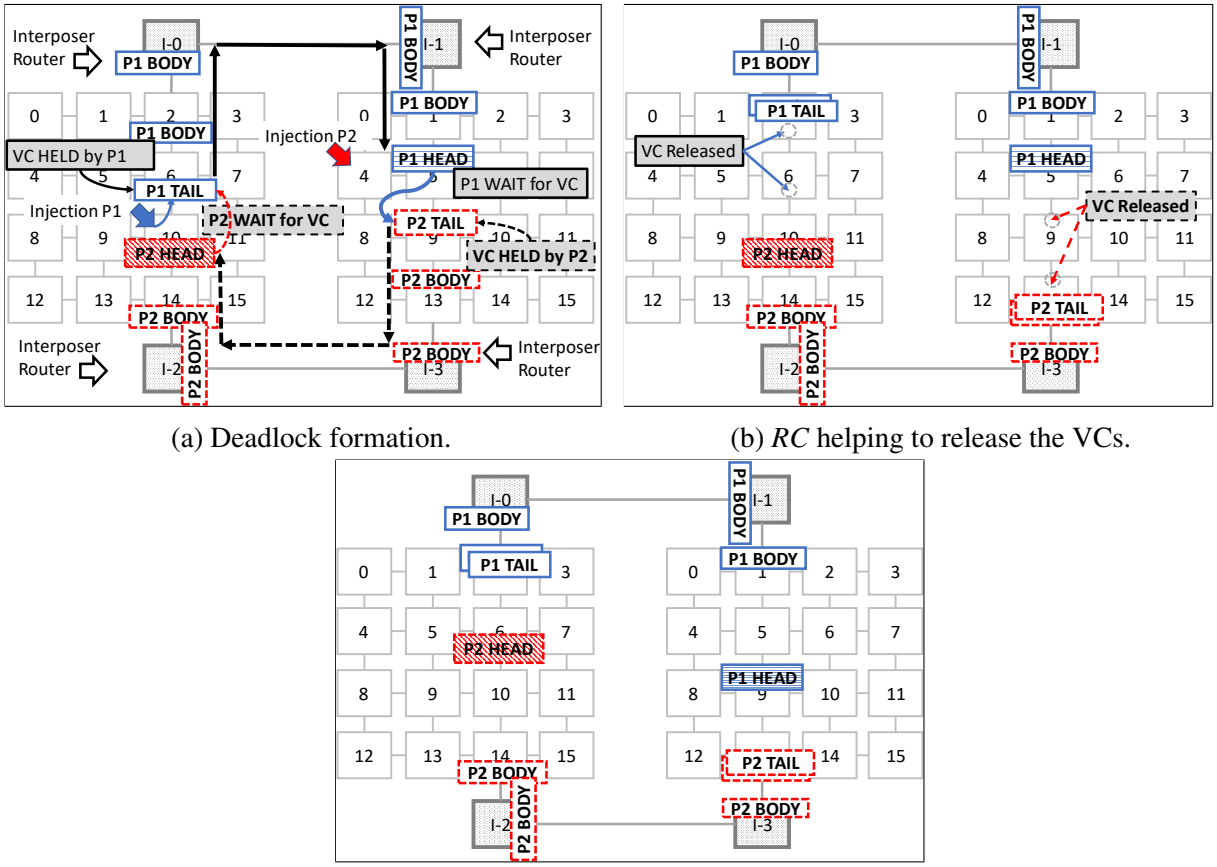
Following recent surge in machine learning assisted hardware policy design and execution led us to integrate reinforcement leaning support as a plug-and-play module. This support is unique for any architectural simulator as they do not intend to do that when designed. Adding this component in the simulation framework imposes a lot of constraints in the design, especially puts a cap on the simulation time, beyond which RL support will not be even feasible in terms of the whole simulation time.

3. A SIMPLE DEADLOCK AVOIDANCE SCHEME FOR MODULAR SYSTEMS-ON-CHIP [1]

Ever increasing performance demand and shrinking in the transistor size together result in complex and dense packing in large chips. That motivates designers to opt for many small specialized hardware modules in a chip to extract maximum performance benefits with relatively lower complexity and cost. These altogether opens up new directions for heterogeneous modular System-on-Chip (SoC) research, where a large system is built by assembling small independently designed chiplets (small chips). We focus on the communication aspect of such SoCs, especially newly observed deadlock among chiplets. Even though deadlock is a classic problem in networks and many solutions are available, the modular SoC design demands customized solutions that preserves the design flexibility for chiplet designers.

In this chapter, we propose *Remote Control (RC)*, a simple routing oblivious deadlock avoidance scheme based on selective injection-control mechanism. Along with guarantee on deadlock freedom, *RC* aims to provide a methodology to make each independently designed chiplet seamlessly integrate in any modular SoCs. We realize the modular SoC in the full system simulation framework and observe that system experiences deadlock while exposed to high workload pressure. In addition, we also observe that the existing turn-restriction based state-of-the-art technique imposes very strict turn restrictions while moving transactions from the chiplets to the interposer network and vice-versa. Hence by alleviating those turn restrictions, we achieve up to 56.34% throughput and 15.49% zero load latency improvements on synthetic traffic and up to 20% speedup on real workloads taken from vast range of benchmark suites, over the state-of-the-art turn restriction based technique applied in modular SoC domain.

[1] Part of the data reported in this chapter is reprinted with permission from “A Simple Deadlock Avoidance Scheme for Modular Systems-on-Chip" by Pritam Majumder, Sungkeun Kim, Jiayi Huang, Ki Hwan Yum, and Eun Jung Kim, IEEE Transactions on Computers, Vol. 70, No. 11, Nov 1 2021, Copyright © 2021 IEEE.



(a) Deadlock formation.

(b) *RC* helping to release the VCs.

(c) Deadlock resolved.

Figure 3.1: An example of deadlock, formed in between two (4×4) mesh chiplets, and how *RC* can avoid the deadlock. (a) Packets P1 (blue solid line) and P2 (red dashed line) forming deadlock. (b) *rc_buffer* in the boundary routers store outbound packets. (c) *RC* avoids the deadlock by allowing all the outbound packets to be stored in the boundary routers until they get credit from downstream interposer routers.

3.1 Overview: Remote Control (RC)

We provide a brief overview of the *RC* by first describing the deadlock problem in the modular SoCs and followed by the motivation behind the *RC* proposal, which shows that the solutions, applied in different other fields for deadlock avoidance are not good enough for providing a good performance in our context.

3.1.1 Problem Description

We use an example to describe the deadlock issue in the modular SoC. Figure 3.1a shows a deadlock case in a modular SoC, where two (4×4) 2D mesh chiplets are connected through an interposer. We denote router i on chiplet j as R-i/C-j for simplicity, where chiplet-0 is on the left and chiplet-1 is on the right. In this system, R-2/C-0, R-14/C-0, R-1/C-1 and R-13/C-1 are boundary routers connected to the interposer network. Packets P1 and P2 are two outbound packets in a circular hold-and-wait situation, forming a deadlock. The P2-head flit in R-10/C-0 requests for the south VC of R-6/C-0, which is held by packet P1. On the other hand, P1-head flit in R-5/C-1 requests for the north VC of R-9/C-1, which is taken by P2. Such a case creates a circular hold-and-wait situation and forms a deadlock, where neither P1 nor P2 can make forward progress[2]. To avoid deadlock in this scenario MTR may impose turn restriction from *R-6/C-0* to *I-0* through *R-2/C-0*, increasing pressure on the other boundary of C-0 for outbound traffic. Note that MTR needs to impose more turn restrictions to avoid all other possible circular hold-and-wait scenarios.

3.1.2 Motivations for *RC*

Limitations in the existing techniques discussed in Chapter 2 motivates us to find a better solution. State-of-the-art MTR identifies an important emerging problem and provides a solution. However, MTR has a few limitations. The major constrain in MTR is that it forces the chiplet designers to implement turn restrictive routing to guarantee deadlock freedom in the modular SoC, which is the main motivation of *RC*. In addition, extra turn restrictions can lead to non-minimal path for intra- and inter-chiplet traffic. Also, the turn restrictions obtained by MTR does not balance the turn restrictions among the boundaries well. Hence, a few boundary routers get huge inter-chiplet traffic load while others do not, causing several hotspots in the system. MTR also constrains the routing design and incurs design overhead. The complexity of the CDG analysis, which is the core of this technique, grows exponentially with the increase in the number of chiplet routers

[2]Progress/forward-progress means moving near to the destination.

	Modularity	Design Efficiency	Energy Efficiency	Performance
MTR	$+++$	$++$	$+++$	$+$
VC-SEP	$--$	$++$	$---$	$-$
ITB	$+++$	$--$	$--$	$++$
RC	$+++$	$+++$	$+++$	$+++$

Table 3.1: Qualitative Comparison with Different Deadlock Avoidance Techniques for Modular SoC. $(+)$ means high and $(-)$ means low. We project the degree of high and low efficiency with number of $(+)$ or $(-)$, respectively.

and boundary routers, which unnecessarily elongates the design cycle. Moreover, MTR imposes restrictions on interposer routing to restrict the inbound packets route, which increases system network design complexity and traffic contention further.

VC-SEP is a well known technique for avoiding deadlock. The main drawback of this approach is that it is very expensive in terms of energy and area consumption. In addition, the buffer utilization is low, which leads to sub-optimal performance. Hence, even though this solution is fairly simple, it is not attractive for designing cost-effective and high throughput modular SoC. Hence, we use the idea of traffic isolation and come up with more efficient implementation by adopting remote injection control mechanism with small buffer.

ITB could be a promising solution for deadlock problems in Modular SoC. However, it has two major drawbacks in this context. (1) Dropping a packet in on-chip reliable network introduces unnecessary complexity and overhead. (2) Ejection and reinjection in multiple nodes increases overall hop counts as well as average packet latency. Furthermore, due to packet dropping/reinjection and use of ACK/NACK packets, the overall system throughput suffers. In principle *RC* is different than this solution as *RC* does not rely on ejection and reinjection for deadlock freedom. In fact, *RC* ensures deadlock freedom just by isolating two types of traffics in the system, which is inspired by the VC-SEP idea.

In Table 3.1, we summarize the comparisons of these techniques and project the expectation of *Remote Control*, which aims for improving the limiting aspects of existing solutions. In a nutshell,

the goal of *RC* is to provide routing design flexibility and eliminate unnecessary packet dropping incurring packet re-transmission by introducing a flow control based technique. Additionally, *RC* targets to save energy and area by segregating traffic only in chiplet boundary routers.

3.1.3 Remote Control

RC is a deadlock avoidance solution for modular SoC, implemented using injection control imposed on outbound packets from nodes connected to non-boundary chiplet-routers. Since outbound packets get consumed in other chiplets, to avoid cross-chiplet deadlocks, we provide intermediate sink (*rc_buffer*) for outbound packets in the boundary routers. Therefore, outbound packets are drained to *rc_buffer* so that they release the chiplet VC buffers to be used by intra-chiplet and inbound packets. If all the chiplets follow the same, intra-chiplet and inbound packets are never indefinitely blocked by outbound packets, and in turn outbound packets can also make progress, as an outbound packet for one chiplet is an inbound for other chiplet. In this section, we first walk through a simple practical example to show a case of deadlock formation between two chiplets and how *RC* can solve it. Then we generalize it for any chiplet-based systems and theoretically prove that *RC* guarantees deadlock freedom in Modular SoCs.

3.2 Deadlock Avoidance using *RC*

As shown in Figure 3.1a, an outbound packet P1 in C-0 is blocking P2 packet to reach its destination while P2 is blocking P1 in C-1. *RC* separates outbound packets from others in the boundary router, and allows them to be stored completely in the *rc_buffer* until they get credits from the downstream interposer router. This makes sure that the chiplet VCs will get free in a bounded time, after the header flit reaches at *rc_buffer*. Hence, P1 will release all the VCs currently blocked in C-0 and so does P2 in C-1. That is why with *RC* the circular channel dependency among chiplets will never result in deadlock.

3.2.1 Deadlock Freedom

We propose the theoretical support for *RC*. The *rc_buffer* reservation is atomic for each of the requesters and allows only one outstanding *rc-request* per requester at any point of time. The

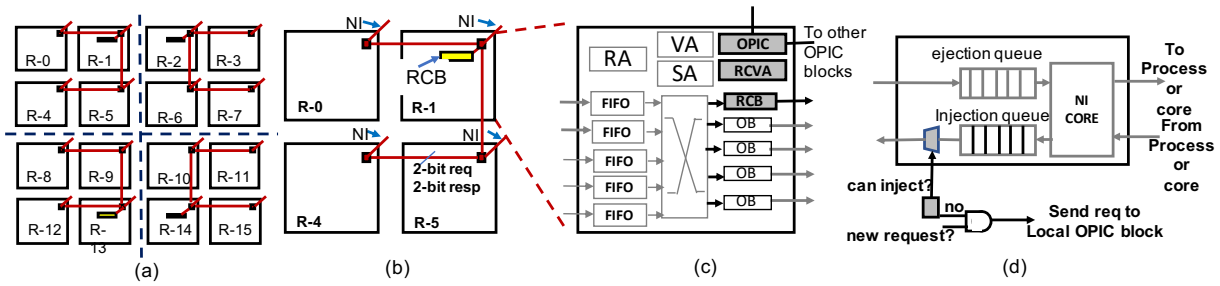


Figure 3.2: Remote Control: (a) Permission network consists of multiple Outbound Packet Injection Control (OPIC) blocks connected in a tree fashion. Each router has one OPIC block. (b) One OPIC tree, and each edge is 2-bit request and response line. (c) Boundary router with the newly added components marked with gray. Note that the router attached to a non-boundary node does not have RCVA and RCB. (d) The changes in NI of the non-boundary router marked in gray. There is no change in NI attached to boundary routers.

request for a slot in the *rc_buffer* for any outbound packet is granted only when there is space for the whole packet, and the slot is reserved. The slot is released once the tail flit leaves the *rc_buffer* and then only a new request is granted. We define one rule and two definitions to keep our theorem statement concise. We layout the proof of the theorem by contradiction, starting by assuming that there is a deadlock.

Injection Rule: An outbound packet can be injected from NI to its attached router **if and only if** the *rc-request* is granted.

Definition-1: *Inflight outbound packet* is the packet whose source and destination are in different chiplets, and holds at least one VC in one chiplet-router. Note that once the outbound packet leaves its source chiplet, it is considered as *inbound* packet for its destination chiplet.

Definition-2: *Destination boundary router* is a chiplet-router used by a set of nodes/routers in the chiplet as a gateway to communicate between the chiplet and interposer. Any communication between these nodes/routers in the chiplet and interposer happen only through their *destination boundary router*.

Theorem: *The SoC network is guaranteed to be deadlock free as long as all the outbound packets reserve slots in rc_buffer in the destination boundary router before injection (Injection Rule).*

Proof: We define the network system (S) at any instant T as $S_T = \{Q \mid Q = \{\lambda, \beta\}, \lambda \subseteq P_T,$ β is the set of all the buffers reserved by packets in λ and $\beta \subseteq B\}$, B is the set of total buffers in the system (independent of T) and P_T is the set of total packets in the system at an instant T. Let us assume there exists an i such that $Q_i = \{\lambda_i, \beta_i\} \in Q$ forms a deadlock. Let B denote a set of buffers in all the boundary routers in the SoC system and ρ denote the set of inflight outbound packets. We categorize all possible scenarios into four cases as follows and prove that *RC* avoids deadlock for any modular SoC network by contradicting our initial assumption that Q_i is in deadlock.

```
if βi ⋂ B == :
   - contradiction; //no deadlock ..............(1)
else:
   if λi ⋂ ρ == :
     - contradiction; //no deadlock ..............(2)
   else:
     if  ∀ ρx ∈ (λi ⋂ ρ) ∃ slot in rc_buffer:
       - contradiction; //no deadlock ............(3)
     else:
       - violation (Injection Rule);//no deadlock.(4)
```

(1) If there is no boundary router buffer involved, that means the deadlock is formed only inside a chiplet or in the interposer, but not involving both. Since a chiplet and the interposer use their own deadlock free routing logic, it is impossible to form deadlock without involving both, which contradicts our initial assumption of Q_i being in deadlock.

(2) If there is no inflight outbound packet involved in Q_i, the circular deadlock chain is incomplete as the packets in λ_i are either intra (source and destination in the same chiplet and chiplet routing is deadlock free) or inbound (outbound packet that left its source chiplet as in Definition-1, so cannot connect with its source in the deadlock chain). So there cannot be any deadlock in Q_i.

(3) If the Injection Rule is followed by each chiplet, then *rc_buffer* will let all the inflight outbound packets to sink in there, allowing intra and inbound packets to reach their destinations

following deadlock-free chiplet/interposer routing. This in turn allows outbound packets to leave their source chiplets and become inbound for their destination chiplets (Definition-1). In addition, deadlock-free chiplet routing guarantees that all the outbound packets reach their destination boundary routers in the source chiplets. There cannot be any deadlock as long as all the packets (intra and inbound) reach their destinations. Therefore, Q_i cannot be in deadlock.

(4) If any inflight outbound packet ρ_x in λ_i that has no *rc_buffer* slot reserved in the boundary router, it violates *RC*'s Injection Rule that all the outbound packets MUST reserve a slot in *rc_buffer* before injection. Therefore, this situation cannot happen. Hence *RC* provides guarantee in deadlock freedom for modular SoCs.

3.2.2 Challenges

There are several implementation challenges *RC* may face. (1) Remote injection control implementation needs to establish an extra channel of communication, which may look structurally similar to the credit channel. The challenging part is to keep the communication overhead as minimum as possible. (2) The *rc_buffer*, situated in the boundary router needs to allow all the inflight outbound packets to be drained from the router VCs. In conventional system, a packet leaves the VC from the upstream router only when the VC in the downstream router has been reserved and the router switch is allocated. Whereas *RC* requires the packets to be drained irrespective of the success in the conventional VC allocation in the boundary router. (3) The *rc_buffer* is common for all types of packets, and hence it might suffer from fragmentation if not taken care properly for different packet sizes.

3.2.3 Routing Oblivious Design

RC is oblivious to both the chiplet routing and the interposer routing. The boundary routers are considered as local destinations for outbound packets in a chiplet. Hence, any routing technique can be used to reach to the boundary routers. Through a boundary router an outbound packet reaches a downstream interposer router, and it follows the interposer routing to reach another local destination in the interposer. Since the interposer takes care of the communication between two

chiplets, one inter-chiplet packet only traverses through its source and destination chiplets. Once the packet enters its destination chiplet, it follows the chiplet routing to reach its destination node.

3.3 Implementation

Is this section, we first discuss one possible implementation of *rc_buffer* (RCB) along with a supporting protocol called RC virtual channel allocation (RCVA). Then, we build a permission network connecting outbound packet injection control (OPIC) blocks. Our goal is to achieve deadlock freedom with complete routing flexibility both in the chiplet and in the system backbone network.

3.3.1 Boundary Routers

Each boundary router has three new components, RCVA, RCB, and OPIC as shown in Figure 3.2 (c). In *RC*, RCVA protocol helps to operate the *rc_buffer*, while the permission network built by connecting OPIC blocks transports permission requests and responses to and from the *rc_buffer*. Altogether, they establish a fine control over the outbound packet injection and its safe transmission to *rc_buffer* to achieve deadlock freedom by alleviating mutual blocking among different traffic types.

3.3.1.1 RCVA

RCVA implements two functionalities. First, it makes sure the outbound packets injected from non-boundary routers do not participate in VC allocation (VA) in the boundary router but directly do switch allocation as shown in Figure 3.3. We bypass the VA stage of the router to save its latency, and when switch allocation is successful, the packet reaches the output port through the crossbar. Then, we push the packet in the reserved slot in RCB.

Second, in each cycle RCVA checks if there is any candidate waiting in RCB for VC allocation. The VC allocation logic in RCVA is much simpler and straightforward than that in the VA, as RCVA deals with only one port. Moreover, since RCB collects all the outbound packets from all the input VCs, VA does not deal with the outport that connects the downstream interposer router. Hence, RCVA does not increase the number of stages in the router for any packets. For outbound packets we consider a different router pipeline, which has same number of router stages as in the

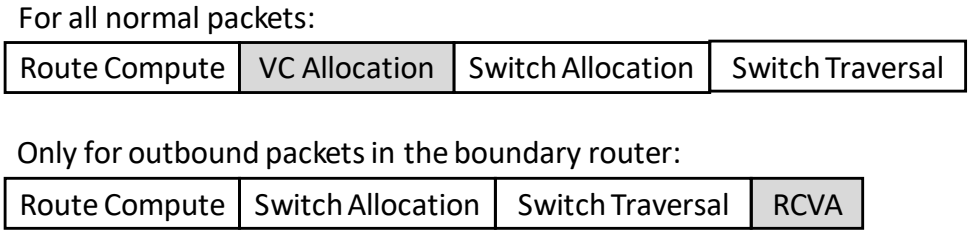


Figure 3.3: Router pipeline stages in case of normal packets followed by modified router pipeline in case of handling outbound packets in the boundary router.

normal router as shown in Figure 3.3.

3.3.1.2 RCB

RCB is a very small buffer located between the crossbar switch and link to the downstream router. It has one or more slots for packets. If the head flit of a packet is written in the first half of a cycle, it may be considered for VC allocation in the other half of the cycle by RCVA. RCB reserves a space when a request arrives from the OPIC block, and allocates one of the empty reserved slots to a packet when its head flit arrives. Flits leave RCB when credit is available for downstream VC buffer. The slot is freed once the tail flit leaves from RCB. To handle different packet sizes efficiently, maximum packet size is being reserved in RCB after receiving request, and once the head flit arrives, we allocate exact number of slots as needed.

3.3.1.3 Permission Network

In this section the basic building block (OPIC) of the permission network is explained, followed by the procedure for building the permission network, maintaining transparency with the chiplet network.

OPIC block: As the name suggests, this block is responsible for regulating the injection of outbound packets in the chiplet through its customized permission network. Each of the blocks supports send-and-receive functionality for both permission requests and responses. In Figure 3.4 we divide the OPIC into smaller blocks, and place them in the timeline with respect to the clock. In the beginning of the clock all the requests and responses reached in the last cycle are registered

Algorithm 1: Build Permission Network

Input : Adjacency matrix **ADJ** of size $N \times N$
Input : Boundary router list **BDRL** of size K
Input : Node list **NODELIST** of size N
Output : list of OPIC trees (**TREES**)

```
1  // initialize the variables;
   Vector   : TREES; // list of OPIC trees;
   Variable: num_trees = # boundary routers;
   Variable: all_visited = false;
2  // initialize one tree for each boundary router;
3  for i ← 0 to num_trees do
       Variable: root = BDRL[i];
4      TREES.push_back(root); // pushed the root of tree;
5  // keep looping until the node_list becomes empty;
6  while all_visited == false do
7      // check for each tree;
8      for t ← 0 to num_trees do
           Variable: cur_tree_size = TREES[t].size();
9          // try to add neighbors for each tree node;
10         for node ← 0 to cur_tree_size do
               Variable: target_node = TREES[t][node];
11             // check for all neighbors of the target_node;
12             for a ← 0 to ADJ[target_node].size() do
                   Variable: nbr = ADJ[target_node][a];
13                 // checking if neighbor of the target_node is there in the node_list;
                   Variable: inList, index, boundary = FindInNodeList (nbr);
14                 if inList == true then
15                     if boundary==false then
16                         // adding node to the OPIC tree;
17                         TREES.push_back(nbr);
18                     // removing the node from node list;
19                     NODELIST.erase(index);
20                     if NODELIST.empty() then
21                         all_visited = true;
```

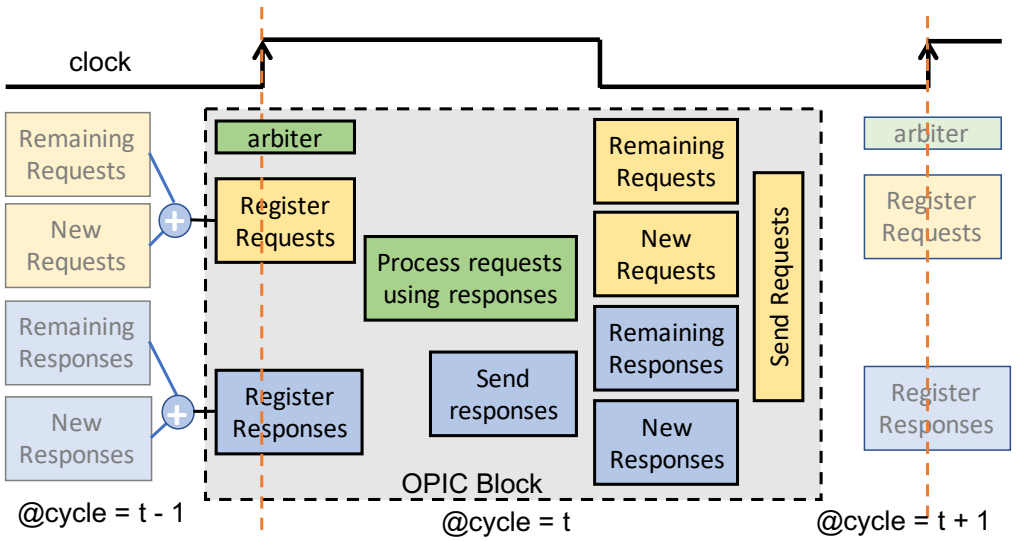


Figure 3.4: Components in an OPIC block along the timeline (not to scale).

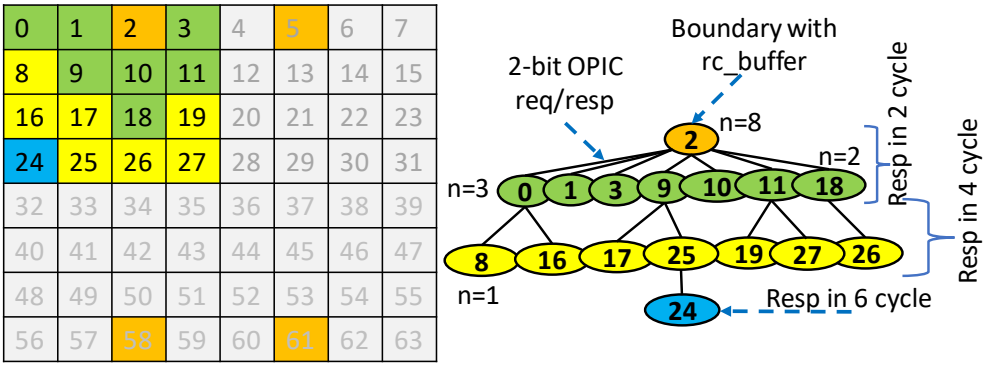


Figure 3.5: Example of permission network in 8×8 mesh with 4 boundary routers with *rc_buffer* connected. n denotes the radix of the OPIC block.

in separate registers for each connecting OPIC blocks. Those registers are read asynchronously and based on the available permissions, responses are sent to requesters in a Round Robin (RR) fashion after processing their requests using combination circuits; after that, remaining requests and responses are also calculated. Again in the next clock, total requests and responses are being registered and the same process continues.

Network: We explain the permission network building process in Algorithm 1 with an example. The chiplet network topology and set of boundary routers being the inputs, this permission

network building process can be demonstrated in general for any chiplet topology and any number or positions of boundary routers. In the example, the input topology is provided for an 8×8 mesh in the form of adjacency matrix, along with boundary router set that contains node numbers 2, 5, 58 and 61. Depending on the technology size, chiplet designers can decide the wire-length for the OPIC block connections (it is a hardware deployment concern, so not included in the algorithm), which may result into different tree depths. The number of permission trees is equal to the number of boundary routers in the chiplet. OPIC block in node-2 connects with OPIC block in nodes-0, 1, 3, 9, 10, 11, 18 using request and response lines of width 2-bit each in an n-ary tree. Similarly, node-0 is connected with node-8, 16 with the request and response lines, and so on.

For instance, at any cycle t, node-8 and node-27 want to inject outbound packets, the requests will be registered in node-0 and node-11 at the beginning of $t+1$, respectively. At $t+2$ the requests from node-0 and node-11 will be registered in node-2. Let us suppose the *rc_buffer* does not have a space, in that case the requests will be standby in these nodes. Now suppose at cycle T one packet space gets free in *rc_buffer* and depending on the arbitration, one of these two will get the response at $T+1$. Suppose node-0 gets the response, then at cycle $T+2$ node-8 will get the permission to inject an outbound packet.

3.3.2 Non-boundary Routers

In a non-boundary router, we append a control on injection process, which allows all the intra-chiplet packets to go without any check. Only for inter-chiplet traffic, the modified injection system checks for injection permission from a local OPIC block. In Figure 3.2 (d), we show that if the permission is not there, the outbound packet is not injected and a request for permission is sent by the local OPIC block to the remote OPIC block in the boundary router through the permission network. Once the response reaches, the outbound packet is injected. Hence, it may happen that if the outbound packet does not get permission to inject, it can block intra-chiplet packets. A separate injection queue for outbound packets in the non-boundary routers may solve the issue. The pros and cons of adding an extra injection queue are discussed as follows.

3.3.2.1 Separate Injection Queue

We consider the separation of injection queues in the non-boundary routers as a design choice for the following reasons. The advantage of having a separate queue can be exploited only if (1) the VC is abundantly available for injection in the router, (2) the workload is very unevenly distributed among chiplets. For instance, in a hypothetical situation with two chiplets, where one chiplet has huge intra-chiplet traffic and huge congestion inside the chiplet, and the other chiplet has a few outbound packets. Those outbound packets may block the intra-chiplet packets in the injection queue for a long time.

In case the number of VC buffers is low/minimal, unavailability of the VCs becomes the bottleneck and injection queue separation turns to be almost irrelevant. So in our general design, we do not consider an extra injection queue for the outbound packets as it increases the NI design complexity significantly.

3.3.3 Case Study: Modular CPU-GPU Integration Using Silicon Interposer

To check the feasibility of our design, we conduct a thorough case study on simple heterogeneous system comprised of four GPU chiplets (16-PEs/GPU, 32-SIMD/PE) and one CPU chiplet (4-cores) [7]. The first challenge we face is to equip the network of each chiplet with *RC* independently. In non-boundary routers the area overhead of OPIC is less than 0.2%. However, in boundary routers area overhead is almost 1.6% over the router area, including *rc_buffer* of depth 8 packets, while the overhead on the NI is negligible. In the established permission network, we ensure the setup and hold time has sufficient slack for both requests and responses between the OPIC blocks. The permission network works independent of the chiplet network, operating with the same 2GHz frequency as that of the chiplet network.

Once the chiplets are equipped independently with *RC*, they are ready to be integrated in an SoC using 2.5D active interposer as shown in Figure 3.6. In this SoC, the interposer provides a (4×4) mesh network for inter-chiplet communication. Since in the interposer network, edge routers are connected with DRAM memory, memory controllers are connected with those routers.

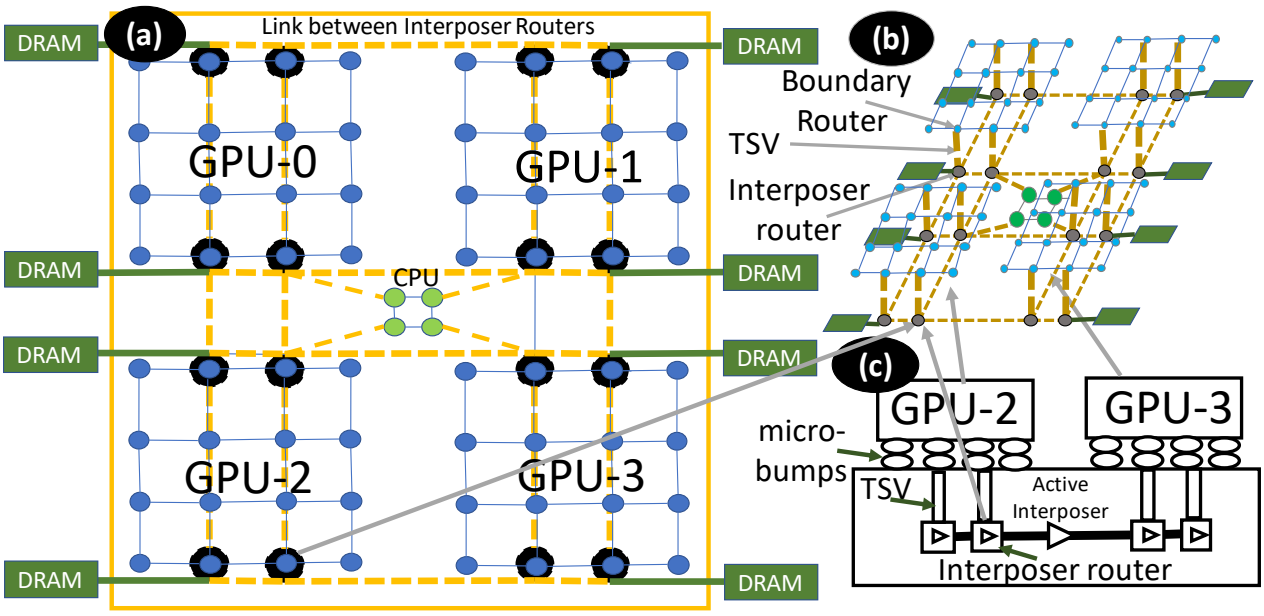


Figure 3.6: SoC viewed from different angles. (a) Shows the top view of the SoC. There are four GPU chiplet (4×4 mesh) at the four corners, and a CPU chiplet (2×2 mesh) in the center. DRAM memory is connected with edge interposer routers. (b) 3D view of the same SoC, highlighting the interposer router, boundary router, and TSV that connects them. It also show the active interposer, and the mesh network in the interposer. (c) Microscopic cross-section view of SoC highlighting the micro-architectural details of 2.5D SoC integration on active interposer.

The edge routers in the interposer also contain the coherence directory. Chiplets are connected with the interposer using micro-bumps. TSV connects the micro-bumps with the interposer routers. Interposer routers use internal link to connect with each other. For instance, in Figure 3.6 (c), if GPU-2 wants to send a request packet to GPU-3, then that packet will reach to the boundary router of GPU-2 first. In the boundary router, the packet will make an entry in *rc_buffer*. From the boundary router, the packet will reach to the interposer router through the TSV. Once the packet reaches to the interposer router, it will be routed to the interposer router that is connected to GPU-3. Again, through the TSV the packet will reach from the interposer router to the boundary router of GPU-3. Figure 3.7 shows the link utilization pattern for this system. We observe that the interposer network experiences much more load as compared to individual chiplets as multiple chiplets are connecting to it and each accesses the memory through this network, and also there are coherence traffic going across chiplets through the interposer network. Based on this observation we propose

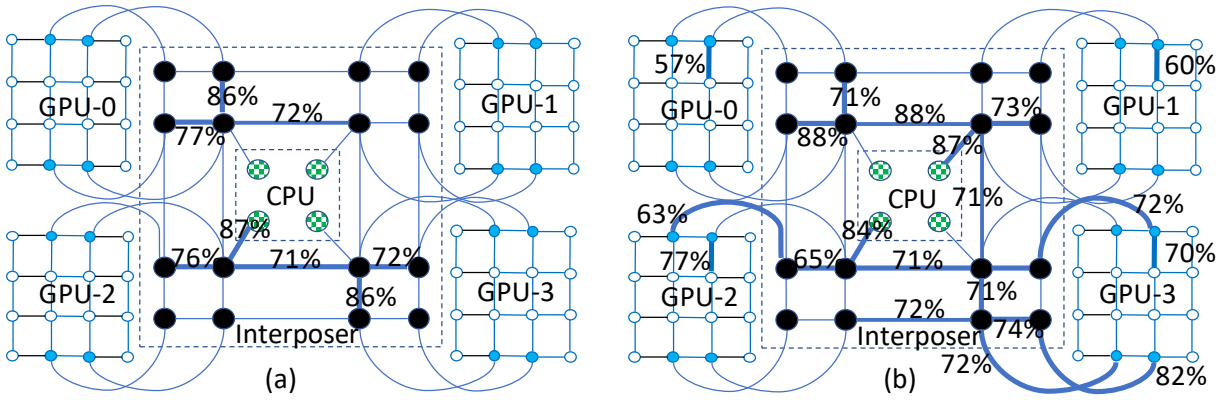


Figure 3.7: Difference in maximum link utilization between (a) *Parking* and (b) *Modular Turn Restriction* technique. Each number on the link represents the percentage of maximum number of cycles the link was busy across all the sample (10,000 cycles) periods.

to enhance the interposer network for sustaining more bandwidth.

3.4 Methodology

We verify and evaluate feasibility of the design in terms of both functional correctness and design efficiency by synthesizing permission network, along with state-of-the-art 4 stage routers RTL [154], using *TSMC 45nm* library. Functional correctness is verified using extensive test-cases. Average area, power, and delay experienced by permission network are analyzed by simulating RTL model.

To evaluate viability of *RC* in the target system, we build software prototype in gem5 [134]. We experience that *RC* can be seamlessly integrated in cycle-accurate network model of BookSim [155]. Hence for full system setup, we integrate BookSim with gem5. We configure gem5 for both heterogeneous (CPU-GPU) and homogeneous (CPU-CPU) systems. We want to prove that *RC* is easy to integrate in full systems and its functionality does not introduce any new issue and report the observed full system performance.

Finally, we thoroughly study the deadlock issue in hierarchical network system using extensively modified BookSim (reliably implement hierarchical network topology and routing) for several synthetic traffic patterns across large range of injection rates (even beyond saturation points), different VC and *rc_buffer* sizes, different number of boundary routers, different network dimen-

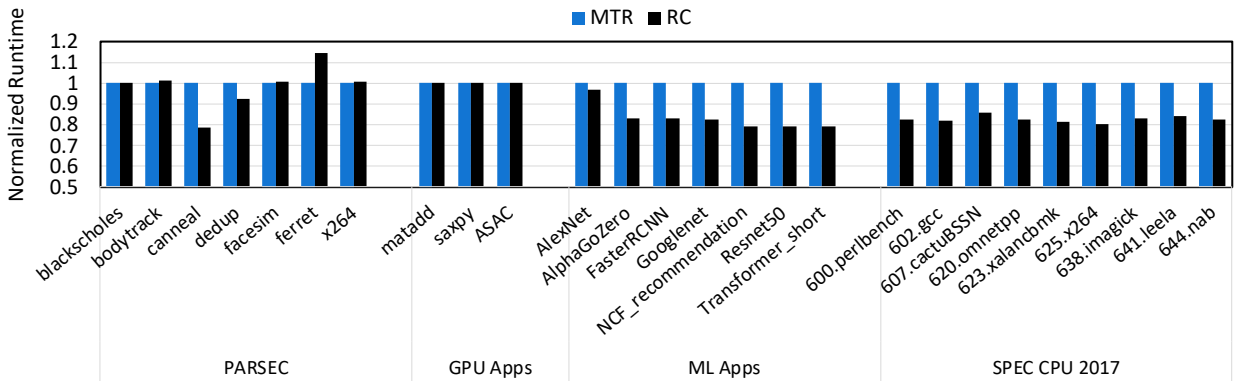


Figure 3.8: Normalized execution time for real applications **(lower is better)**.

Parameter	Value
CPU	2 GHz frequency, TimingSimple
CPU Cache	L1I and L1D - 32KB 4-way L2 - 64KB 8-way
GPU	1 GHz frequency [156]
GPU Cache	SQC (shared L1I) - 32KB, 8 way TCP (private L1D) - 16KB , 16 way TCC (Texture Cache per Channel) - 256 KB, 16 way
Memory	Build-in memory model in Gem5 [157]
Network	Booksim integraded with Gem5, 4-stage routers 1-flit buffers per control VC, 4-flit buffers per data VC 64 bit flit size and channel width
Permission N/W	2 cycles/ OPIC hop (round trip including OPIC block latency)

Table 3.2: Parameters of simulated architecture.

sions, and different routing techniques. We want to prove that in modular SoC, routing flexibility is not an option, it is necessity.

3.4.1 Experimental Setup

In full system setup, we integrate BookSim with Gem5 to simulate network of Compute Units (CUs) in the GPU chiplets (GCN-3 [156]), CPUs in CPU chiplet, and also different chiplets on active interposer as summarized in Table 4.3. We use 4 stage routers having 1-flit buffers per control VC and 4-flit buffers per data VC. Flit size and link channel width is 64 bit. The control

packets are 1 flit and data packets are 5 flits. In homogeneous setup we configure SoC using multiple (4×4 mesh) CPU chiplets only and use MOESI hammer as the coherence protocol. Heterogeneous setup uses the multi-chiplet APU configuration [7], consisting of four GPU chiplets (4×4 mesh, 16 CUs), one CPU chiplet (2×2), and an active interposer (4×4 mesh) as shown in Figure 3.6. We use the in-built memory model in gem5 equipped with eight memory channels and 8 banks per channel. We run heterogeneous system-level simulation on APU applications taken from AMD ROCm Developer Tools [158] and Rodinia [159] suites. We also evaluate *RC* in homogeneous full system setup using PARSEC [160] and SPEC CPU2017 [161]. For running the Machine Learning applications, we attach one accelerator [162] on each node and experimented for training and ring all-reduce operations using underlying BookSim network simulator. Unless otherwise mentioned, for synthetic experiments packet size is 8 flits; we use four 4×4 chiplet and one 2×2 chiplet connected using 4×4 interposer network, having 2-VC-4-stage routers with 4-flit buffer depth and 4 packet space in the *rc_buffer*.

3.4.2 Traffic Patterns

3.4.2.1 Synthetic Traffic Patterns

In Figure 3.9 we show the pictorial representation of the synthetic traffic patterns, assuming source-0 to source-63 on the Y-axis from top to bottom. In the X-axis we have the dest-0 to dest-63 from left to right. For instance in *bit-complement* has $zero$ intra-chiplet traffic, as the source and destination in this traffic pattern lied in different chiplets. The node number distribution for the chiplets is as follows. Node-0 to node-15 (chiplet-0), node-16 to node-31 (chiplet-1), node-32 to node-47 (chiplet-2), node-48 to node-63 (chiplet-3), node-64 to node-68 (chiplet-4). Similarly for other traffic patterns also, source and destinations are divided into chiplets. In Figure 3.10 we show the percentage of inter-, and intra- chiplet traffic for each synthetic traffic patterns.

3.4.2.2 Application Traffic Pattern

The applications run on system level setup as already described. In general most of the communications happen among edge interposer routers and CPUs/GPUs. These interposer routers

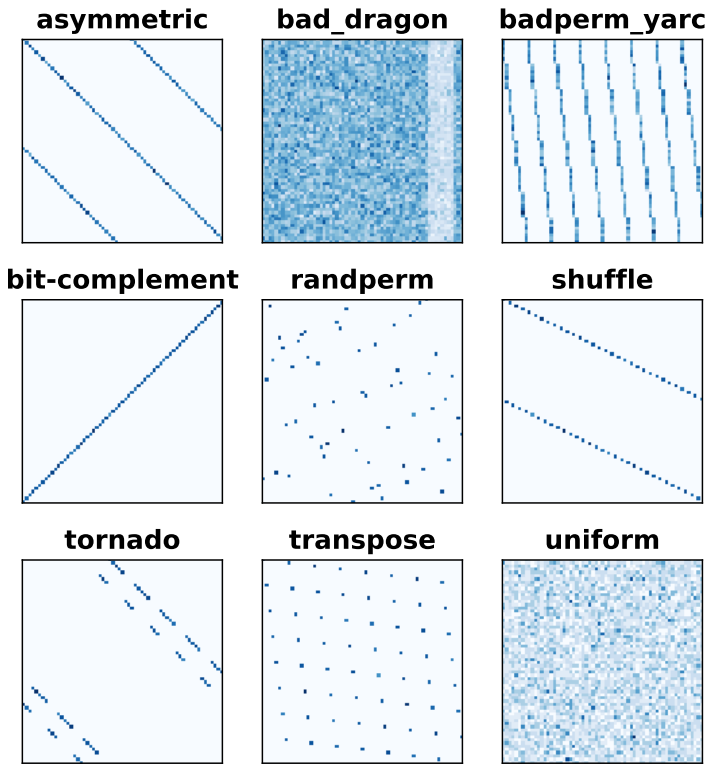


Figure 3.9: Pictorial representation of synthetic traffic patterns. These graphs are drawn by collecting traffic (sources and destinations) at runtime.

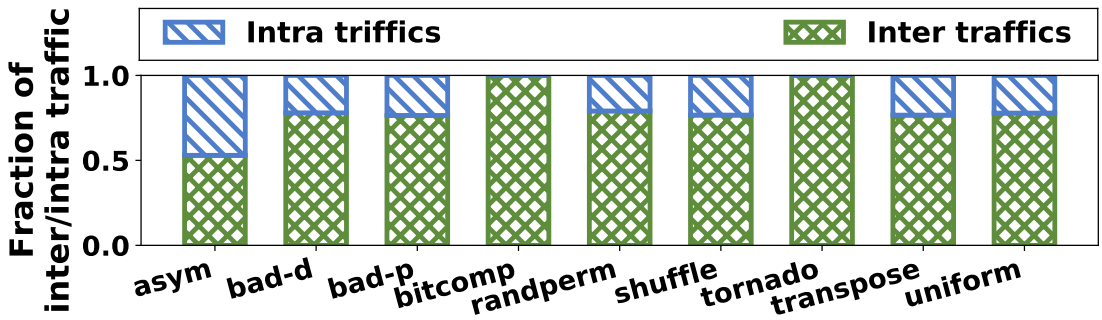


Figure 3.10: Fraction of inter/intra-chiplet traffic.

contain memory controller, and coherence directory. Since in our configuration, CPU and GPU share memory space, huge number of the communications happen either between CPU and memory controller, or between CPU and coherence directory. Also, significant communications happen in between the GPUs and memory controllers, or GPUs and directory controllers. There are very low communication between GPU chiplets, and absolutely no direct communication between CPU and GPU chiplet. CPU and GPU they communicate with each other using shared memory. In Figure 3.7 we show one example of link utilization for *Sync/AsyncAC* application for both *Parking* and *Modular Turn Restriction*. For both the techniques the interposer links are highly used for regular memory accesses as well as for offloading job from CPU to GPUs, and getting back results in the CPU through main-memory, connected with edge interposer routers. Interestingly for *Modular Turn Restriction* we observe heavy link utilization in the boundary routers of the GPU chiplets.

3.4.3 System Speedup

We evaluate our design using both latency sensitive workloads as well as throughput sensitive workloads as shown in Figure 3.8. We evaluate our design for the latency sensitive PARSEC benchmarks and GPU benchmarks and observe performance difference with *canneal* in which the average packet injection rate is moderate to high and almost similar performance for other benchmarks with MTR as *RC* performs also similar to MTR for low and moderate network load. On the other hand for throughput-sensitive programs, we expose both the techniques to several Machine Learning applications [162] as well as parallel execution of multiple instances (32 copies of

same application on 32 different cores) of SPEC CPU2017 benchmarks [161]. As expected, we observe 14% to 20% system speedup, only attributed to an efficient deadlock avoidance scheme (*RC*). Among the throughput hungry benchmarks, *AlexNet* is a slight outlier as the amount of time spent on communication for this benchmark is significantly low as compared to other benchmarks. In addition, full system performance can be boosted by prioritizing delinquent packets [163]. We can partially achieve that by simply modifying the arbitration policy in RCVA. Since, that is orthogonal to our current work, we leave the performance optimization for the full system setup as our future work.

3.5 Performance Evaluation

Keeping the gravity of the inter-chiplet deadlock problem in mind, we quantitatively compare *RC* with MTR, ITB and VC-SEP. We extensively modify BookSim to reflect the hierarchy of networks. We introduce the concept of chiplet and interposer in BookSim to reflect their independent topology and routing. Different link latencies are also reflected depending on their length in the interposer. Initial system we mimic in BookSim, is similar to our prototype with CPU-GPU in terms of their corresponding network. Then we expand our design space to evaluate different sensitivity aspects for complete study. To thoroughly study the deadlock formation we use several synthetic traffic patterns across huge range of injection rates. We observe that since the chiplets and interposer routing techniques are deadlock free, inter-chiplet deadlock forms during high congestion, near to the saturation points.

3.5.1 Throughput Analysis

Figure 3.11 shows that *RC* outperforms MTR, ITB, and VC-SEP in terms of network throughput in all the synthetic traffic patterns. We explain the throughput for uniform random (UR) traffic as a representative of synthetic traffic patterns.

In *UR*, the source and destinations are generated randomly, where most traffics result into inter-chiplet communication. For example, 3/4 of the generated traffics consist of outbound packets in the simulated configuration, which poses more stress on the boundary routers and interposer

network. Across all the techniques, VC-SEP has least throughput, due to under utilization of buffer resources. While in MTR, turn restrictions on boundary routers create load imbalance, leading to throughput degradation. In ITB, ACK/NACKs packets are used for re-transmit request, data and control packets, which leads to higher network load and saturates the network earlier. In addition, ejection and reinjection of packets add latency in the critical path. In contrast, *RC* regulates outbound packet injections facilitated by *rc_buffer* in boundary routers and frees VC usage constraints for better resource utilization. Additionally, *RC* provides routing flexibility so that traffics can be distributed evenly to all boundary routers. With these benefits, *RC* improves network throughput upto 1.7×.

To analyze the traffic distributions and communication bottlenecks of different designs, we depict hotspots as heatmap for UR for MTR, ITB, VC-SEP, and *RC* at their near saturation load[3] as shown in Figure 3.12. Hotspot is defined as the average packet residency time in the router. Darker color represents higher packet residency time due to congestion. MTR imposes multiple extra turn-restrictions, resulting into hotspots due to imbalanced traffic distribution inside chiplets as shown in Figure 3.12a, which leads to low network throughput as shown in Figure 3.11. Heatmap for VC-SEP, as depicted in Figure 3.12c, shows the severe congestion throughout the SoC network, which is due to the intensive usage of limited outbound VCs, making network saturation early. Interestingly, for ITB the contention inside the chiplets is very low, which can be attributed to packet drop [11] that yields the buffer resources in the network. However, extra packet transmissions cause high energy and power consumption in the chiplets. Since *RC* has uniform flow of packets as shown in Figure 3.12d, it exhibits a better throughput than MTR and VC-SEP. *RC* alleviates the long waiting of outbound packets from the chiplet routers. However, the contention in the interposer network partially offsets the throughput benefit, observed using *RC*. Note that we have plotted the heatmap with different injection rate (throughput injection rate) for each technique to show their distinct saturation behaviors, and point out the key reason for saturation. We notice that across all the techniques the interposer network is heavily used (outbound packets from mul-

[3]Different techniques have different saturation load.

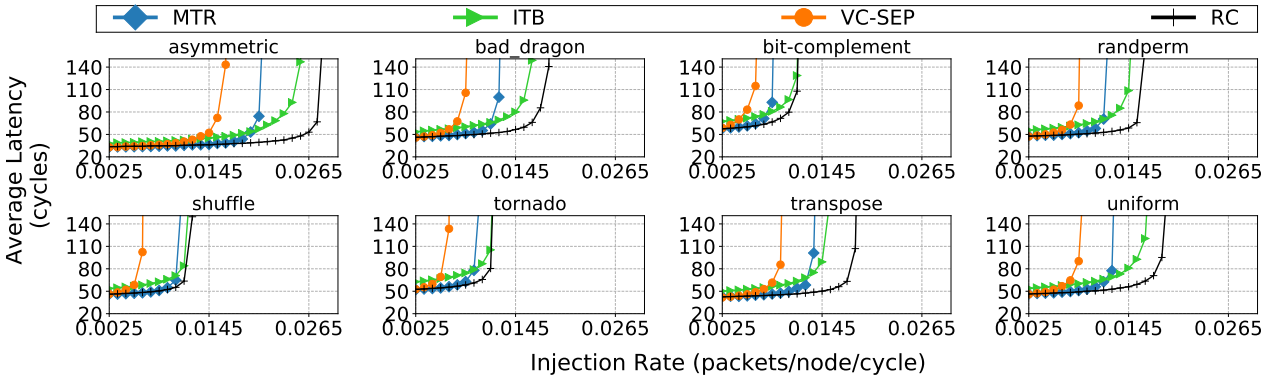


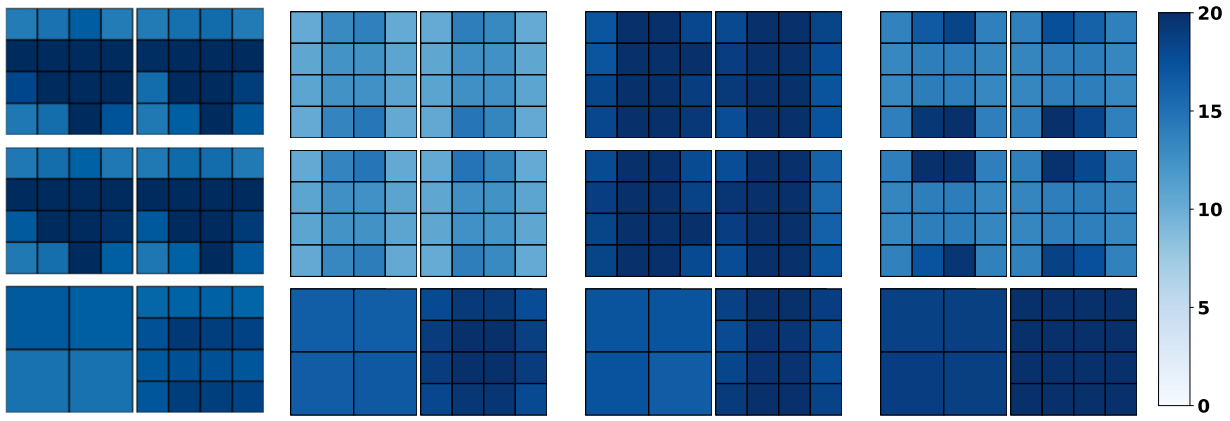
Figure 3.11: Throughput graph for synthetic traffic pattern study. (#VC = 2, VC buffer size = 4, packet size = 8 flits, *rc_buff* = 4 packets).

tiple chiplet nodes go through one interposer network), which could be a bottleneck for achieving throughout improvements. To alleviate the contention from the interposer, we plan to extend our work to investigate innovative SoC topologies as our future work.

3.5.2 Latency Analysis

As shown in Figure 3.11, we observe a similar low load latency among *RC*, MTR and VC-SEP across various traffic patterns. Figure 3.13a presents the detailed comparison of low-load latency for *UR* as an example. It shows ITB increases a few more cycles as compared to other techniques. Extra ejection and re-injection at low-load incurs two extra hops that causes high latency overhead. On the other hand, MTR fails to follow minimum path to destination because of the extra turn restrictions. In case of *RC*, because of the modular design, route is optimized in each of the independently designed modules, which may not result into shortest path from source node to destination node. We expect to achieve better low-load latency for *RC* if we incorporate more chiplet information by relaxing modularity constraints while designing interposer routing.

In Figure 3.13b, we show the average packet latency breakdown for *RC* for UR to understand the overhead incurred by injection control. It shows that the portion of granting delay for *rc_buffer* reservation over packet latency increases with the increase in network load at the beginning, and decreases while moving from medium load to high load. This is because at low load, *rc_buffer*



(a) MTR. (b) ITB. (c) VC-SEP. (d) RC.

Figure 3.12: Heat map of average packet residency latency (cycles) per router (smallest cube) in UR plotted for **near saturation point** for each technique. Top four cubes (big cubes) represent GPU chiplets, having 16 routers (small cubes) each. The bottom-left cube is the CPU chiplet having 4 routers. Beside the CPU chiplet we show silicon interposer with 16 interposer routers.

reservation causes constant delay without contention. Whereas at medium load, contention on *rc_buffers* increases the granting delay. As injection rate increases to high load, the exponential injection queuing delay dominates the packet latency, which reduces the impact of *rc_buffer* reservation significantly. To alleviate round-trip delay of the permission network, *rc_buffers* can be operated in a more proactive way, similar to token circulation rather than on-demand requesting to reduce the constant delay at low to medium load. That may improve the permission request-response delay, if enough tokens flow throughout the network [164].

3.5.3 Routing Obliviousness

In this section, we show *RC* is routing oblivious by implementing Dynamic Credit-based Routing, where each router adaptively selects either XY, or YX routing depending on the credit availability in the downstream router. To demonstrate the benefits of routing oblivious *RC*, we alleviate the bottleneck in interposer as discussed in Section 3.5.1 by providing 2 extra VCs only for interposer routers. In Figure 3.14, we show that when dynamic routing is applied, the throughput improves in both smaller system (68 node, 84 routers) and bigger system (272 nodes, 304 routers) by 15.3% and 21%, respectively. The main advantage of *RC* is that it gives complete freedom to

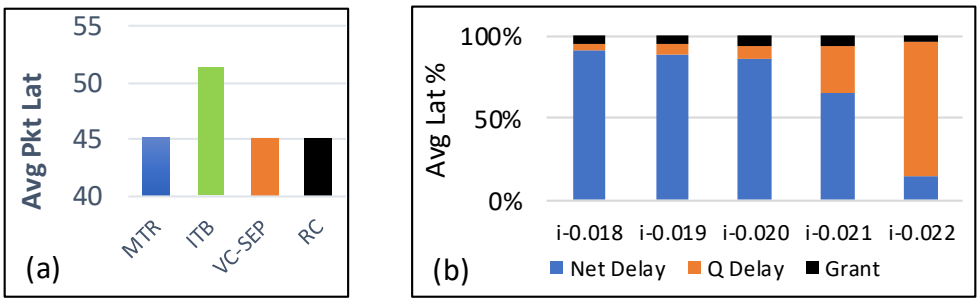


Figure 3.13: Analysis on throughput graph in Figure 3.11. (a) Zero load latency of UR, representing the trend for others as well. (b) The network breakdown for *RC* across different injection rates for UR. “Net Delay" is the network delay including retry latencies. “Q Delay" is the injection queue latency. “Grant" response latency for getting the permission from OPIC including waiting for *rc_buffer* full condition.

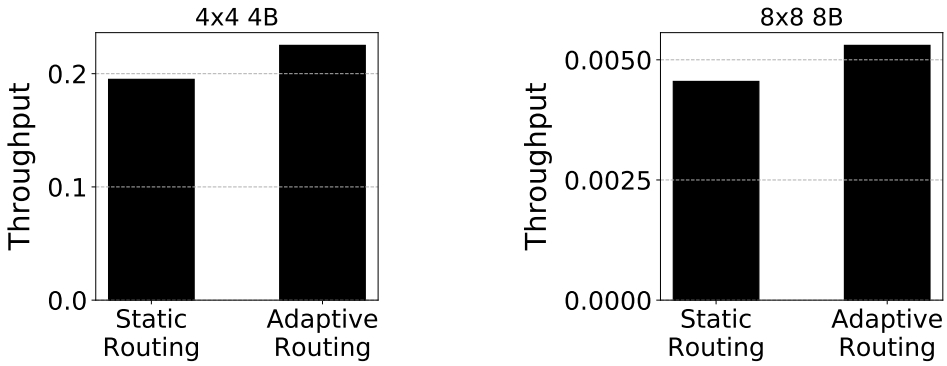


Figure 3.14: Throughput improvement by using adaptive routing for different size of modular SoCs for the following systems. The left two bars are for 4 boundaries in 4×4 GPU chiplet and 4 boundaries in 2×2 CPU chiplet. The right two bars 8 boundaries in 8×8 GPU chiplet and 4 boundaries in 4×4 CPU chiplet.

the chiplet designers to implement the best routing for the chiplet, using their domain expertise, without being worried about system-level deadlock issue.

3.5.4 Starvation and Fairness

The system ensures that starvation never happens by serving each of the nodes in a Round Robin (RR) fashion. In terms of the OPIC delay, the first time request for outbound packet injection is logically decoupled with the consecutive retries, by registering the OPIC response in the local node’s NI. This system works since *RC* proactively responds to the requesters whenever their turn

comes in RR and the slot for entire packet is available.

The fairness issue can be broken down into two distinct situations, (1) when the *rc_buffer* is full, and (2) when the *rc_buffer* is available. When the *rc_buffer* is full, none of the requesters get served, regardless of their location with respect to the *rc_buffer*. In that case, all the requests wait in the OPIC block of the boundary router, so the Round Robin policy can serve them fairly. When the *rc_buffer* is available, a nearer node stands a higher chance to get served than a farther node, if they generate requests at the same time. However, it is not always true. In the OPIC block, when a requester's turn comes, the request gets served following Round Robin policy. This process continues until all the responses present in that OPIC block get exhausted. The serving starts again from the point, where it stopped last time. That is why in some scenarios, even if the request from the near node reaches first, the responses may get exhausted before its turn comes. By the time new response arrives, the request from the far node may get registered. Then depending on the last serving location, either the near or the far requester may get served. In summary, we guarantee that there is no starvation in the system. However, fairness is always not preserved, as the nodes that are near to the boundary router may consume higher OPIC bandwidth than the nodes situated farther, which is an inherent nature of any multi-hop network.

3.5.5 Sensitivity Analysis

We scrutinize the system using various size and number of chiplets to obtain better understanding about system scalability with *RC*. Difference of throughput is also observed with different VC sizes and increasing size of *rc_buffer*. We intend to provide enough insight for estimating the best combination of these parameters for the system designers.

3.5.5.1 System Scalability

We extensively study the system scalability as shown in Figure 3.15 by increasing number of chiplets in Figure 3.15b (132 nodes) as compared to Figure 3.11 (68 nodes). To compare the scalability with different size of chiplets, we also keep the total number of nodes same between Figure 3.15a (132 nodes) and Figure 3.15b (132 nodes) and contrast their zero load latency and

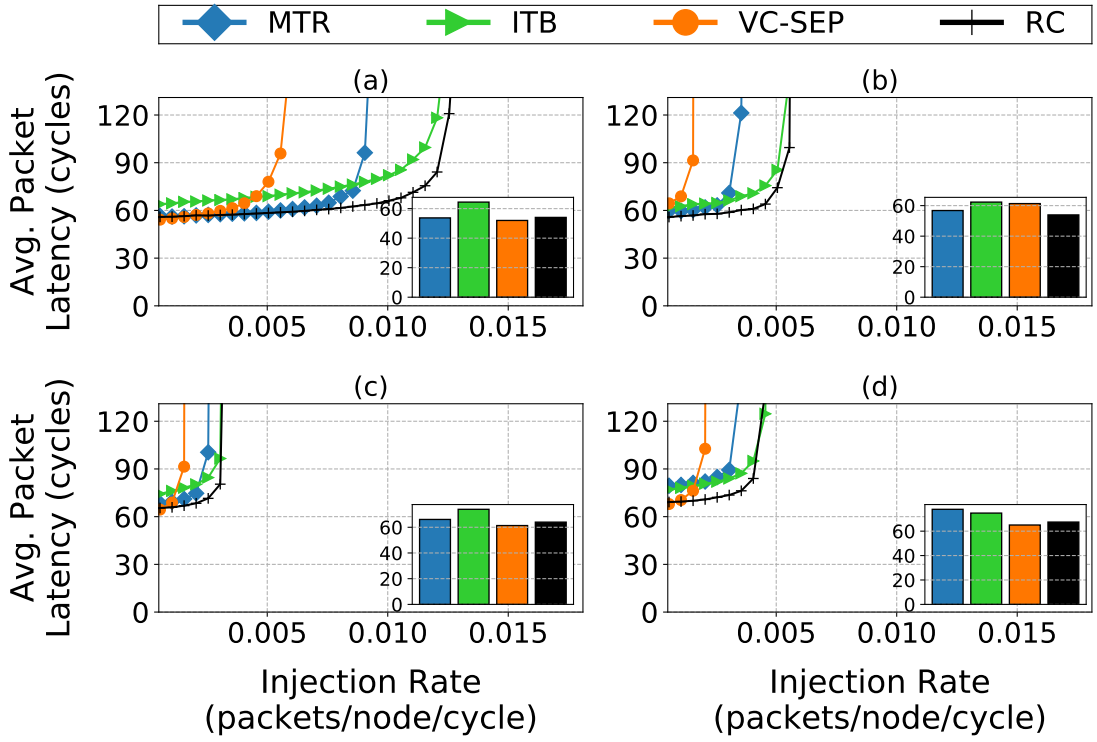


Figure 3.15: Sensitivity study: Scaling up the chiplet size, while keeping the number of boundaries same as 4/chiplet. (a) Eight GPU chiplets of size 4x4 and one CPU chiplet of size 2x2 mesh. (b) Two GPU chiplets of size 8x8 and one CPU chiplet of size 2x2 mesh. (c) Four 8x8 GPU and one 4x4 CPU. (d) Same as (c) except 8 boundaries/GPU chiplet. The small bar chart in each of the graphs represents zero load latency for that particular configuration.

throughput. Figure 3.15d shows a large system with large number of nodes per chiplet (total 272 nodes) with doubled number of boundaries in each chiplet. In all the configurations *RC* outperforms MTR, ITB, and VC-SEP in terms of throughput. Also in terms of zero load latency *RC* exhibits same or better than MTR and much better than ITB. This is because the detour caused by turn restrictions in MTR surpluses *rc_buffer* request delay in *RC*. For example, in Figure 3.15c, MTR has 2 extra hops than *RC*, which accounts for 17% more in average hops. We observe that the throughput difference reduces with the increase in the system size, as more nodes saturate the bisection bandwidth earlier.

We quantitatively show that with the increase in the chiplet size, overhead of OPIC does not hamper performance. In a 4×4 mesh with four boundaries, each boundary gets three requesters,

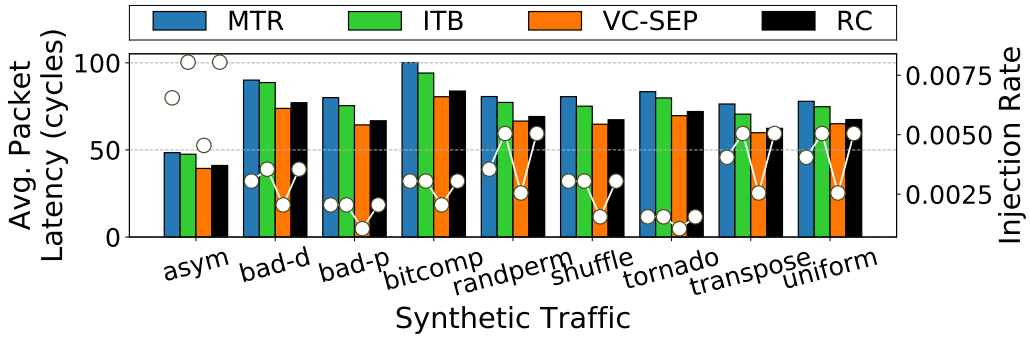


Figure 3.16: Doubled the number of boundaries 8 boundaries/ 8×8 GPU chiplet and 4 boundaries in 4×4 CPU chiplet. The major Y-axis corresponds to zero load latency shown in bar graphs, and minor Y-axis corresponds to the throughput as shown in white dots.

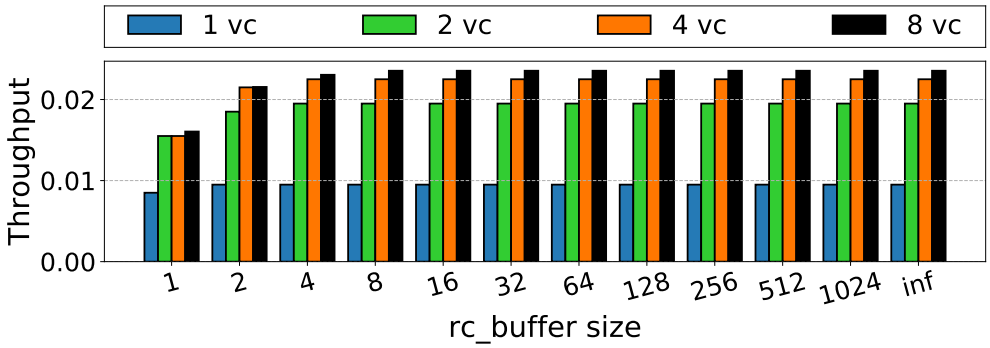


Figure 3.17: Throughput sensitivity and interplay between virtual channel and *rc_buffer* size for 4×4 chiplets (68 nodes setup) with 4 boundaries/chiplet.

and all of them get response from the boundary OPIC block in 2 cycle. However, in a 8×8 mesh network, maximum seven requester nodes can be connected as they are in one, or two hop distance from the boundary. In that case the furthest node from the boundary gets the response in total 6 cycles. Other nodes get response in much lesser time. Since the requests get registered in the next OPIC block, requester needs to send request only once. When we scale the number of nodes further we do not need to reconsider the setup time and hold time, as the amount of work needed to be done in one cycle will still be same. Only the number of cycles of getting response will increase with the increase in distance from the boundary router. However, it is worth noting that we opt for modular SoC design as we do not want to make large chips, rather want to put multiple small

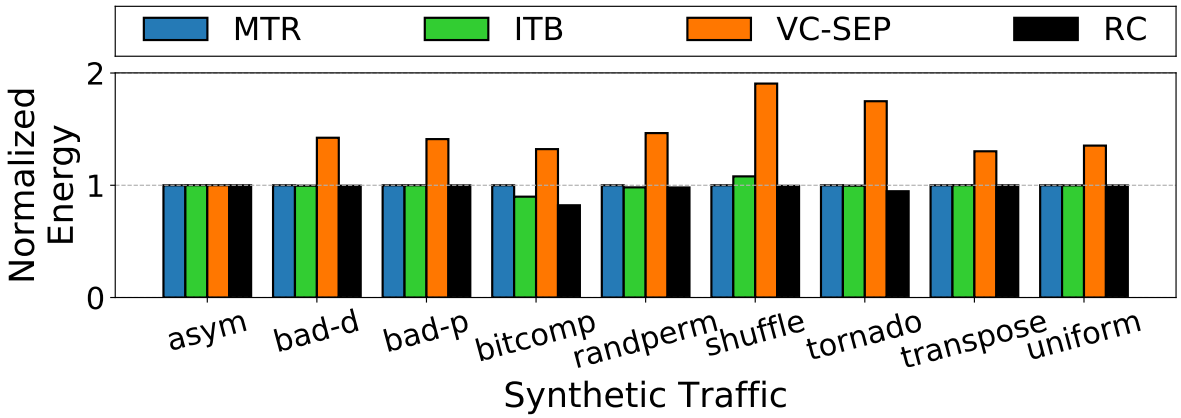


Figure 3.18: Normalized energy for all the techniques, across all the synthetic traffic patterns.

chiplets together to scale the system size. In Figure 3.15a and in Figure 3.15b, we show that system with multiple smaller chiplets in Figure 3.15a has better throughput than system with fewer large chiplets. One reason for the difference in lower throughput with large number of smaller chiplets is the number of boundary routers are same for each chiplet both the systems [7], resulting in more boundaries in total, in case of systems with smaller chiplets.

Going one step further we doubled the number of boundary routers for 8×8 chiplets keeping every other parameter values same. In Figure 3.16, the result shows that the average packet latency in case of *RC* improve significantly) over MTR by up to 17%. Figure 3.15(c) and 3.15(d) show results for SoCs with 4 boundaries and 8 boundaries per GPU chiplet, respectively. Average packet latency increased from 68 cycles to 80 cycles in case of MTR, and there is almost no change for our techniques. Our experimental results show that MTR travels more than 19.5% extra hops as compared to *RC*. This can be attributed to extra turn restrictions imposed by MTR. In addition, since the complexity of CDG analysis increased exponentially, we run the MTR algorithm for 7 days to explore the design space and pick the optimal result, which may not be the optimum turn restrictions for 8 boundary router setup. Interestingly, VC-SEP shows the zero load latency in this setup. However, the throughput suffer a lot because of low VC buffer utilization. In contrast, CDG analysis in 4 boundary setup takes only less than 2 hours to finish in one *intel core-i7* processor.

3.5.5.2 Sensitivity to rc_buffer Size and VC Size

In Figure 3.17, we show impact of *rc_buffer* size on network throughput, which is the saturation injection rate for SoC network with four 8×8 GPU chiplets and one 4×4 CPU chiplet (272 nodes), which shows similar trend for the smaller baseline setup with 4×4 GPU chiplets. We observe that increase of both *rc_buffer* size and the number of VCs have impact on the system throughput. With *rc_buffer* size of 1, we see hardly any throughput improvement with increasing VC sizes. The throughput improvement from single packet slots to two packet slots in *rc_buffer* is almost 2×. Also with 1-VC, increase in the *RC* size improves throughput marginally. Result shows that for all the VC sizes, *rc_buffer* size of 4 is good enough to provide achievable throughput, which is the case in infinite *rc_buffer*, where the OPIC delay is zero. In addition, in terms of throughput, the difference between 4-VC and 8-VC result is also not very significant. Even 2-VC result also shows a good trade-off between throughput and energy consumption.

3.5.6 Area and Energy Analysis

The hardware complexity and area overhead of *RC* is very minimal. As per our detailed synthesis report, in each router of size 49667.53 μm^2, OPIC logic consumes only $785.68\mu m^2$ area, which is 1.6% of the router area. There are four *rc_buffer* in each chiplet, and each has 4 packet buffers, consuming $6.0424\mu m^2$ in total. Area overhead and hardware complexity incurred is negligible as compared to the total chiplet area and complexity.

Since we focus on the network deadlock aspect in this work, we estimate only the network energy to compare between MTR, ITB, VC-SEP and *RC* using DSENT [165] and *rc_buffer* access energy from RTL simulation ($0.10425pJ/flit/access$). The energy consumed by the wire connections in the OPIC tree are not significant. Figure 3.18 shows energy consumption of different techniques normalized to MTR under 0.013 packets/node/cycle injection rate for 100000 packets. It shows *RC*, MTR and ITB consume similar energy across all the synthetic traffic patterns. In contrast, VC-SEP consumes more energy due to low utilization of VCs, leading to longer simulation time that consumes more static energy. We expect *RC* to save more energy by reducing the static

energy in high load since it sustains higher throughput.

3.6 Related Works

Deadlock avoidance mechanisms fan out in two distinct branches, namely VC and turn model based, and flow control based techniques. The first type either rely on turn restrictions, or on dedicated/ordered VC buffer for different traffic types/directions. On the other hand, flow control techniques either control the injection of packets, or ensures bubble in the buffer to avoid deadlocks. The state-of-the-art solves the new SoC deadlock issue using routing based turn restrictive technique while *RC* follows flow control based deadlock avoidance.

3.6.1 VC and Turn Model Based

Duato proposed escape-VC [8], a theory for deadlock freedom for routing with cyclic channel dependency. Duato's theory can be applied for both deadlock avoidance [53, 54] and deadlock prevention [55, 56] techniques. Idea of escape channel cannot be applied directly in modular SoC as the packets in the escape-VC must be propagated using a deterministic deadlock free algorithm, which cannot be guaranteed in a modular SoC. Recently Ebrahimi et al. [57] propose *EbDa* that provides exclusive sets of VCs to isolate traffics (say, intra-chiplet traffic, and inter-chiplet, or outbound traffic) to avoid deadlocks. However, VC separation leads to lower utilization and is shown less attractive in MTR [7], and we also find the same way.

Dally et al. [6] propose to use two or more VCs in order to avoid the cyclic channel dependencies. It ensures deadlock freedom by using total ordering of VCs. Even though this condition is sufficient to avoid the deadlock, it is not necessary [9]. Extra VCs result in increase in the router area and energy consumption. Based on Dally's theory, a few other techniques have been proposed that use additional VCs [59, 60] to avoid deadlock. Another way to achieve strict order of reservation for the shared VCs is by imposing turn restrictions [62, 63] on the packet traversal.

3.6.2 Flow Control Based

For providing deadlock freedom, flow control techniques either regulate the injection [64] of the packets or allow a packet to go forward depending on the buffer occupancy [66] in the ring. The

second concept is coined as bubble flow control by Puente et al. [67] and applied in torus network for the flow control in escape channel. This concept is being used in in-transit buffer for avoiding deadlock in k-ary n-cube torus network [68], and extended later for irregular off-chip network [11], worm-whole switching [70], torus cache-coherent NoCs [71].

Recently Ramrakhani et al. [9] propose *SPIN*, a synchronized flow control technique for deadlock prevention in flat network. It is very challenging to apply synchronized flow control in modular SoC, where the chiplets are designed independently, and connected through the interposer routers. Moreover, synchronization of packet movement among chiplets make the design very complicated.

3.7 Summary

Chiplet-based system integration on an active interposer is a scalable and economic solution for improving system performance. As deadlock freedom is one of the main concerns, we propose *RC*, a simple routing oblivious technique for modular SoCs. It completely protects the idea of modular design by providing total independence to the chiplet vendors, in terms of routing logic, topology, dimension, etc. The low load latency improvements of *RC* over MTR, ITB and VC-SEP are up to 15.49%, 19.17%, and 13.76% across different configurations for all the synthetic workloads, respectively. The throughput improvements achieved by *RC* over MTR, ITB, and VC-SEP are up to 56.34%, 12.12%, and 2.5×, respectively. In full system simulations for real workloads, we improve performance upto 20% as compared to state-of-the-art MTR. As part of future work, we want to investigate application-aware *OPIC* system, where critical packets can be prioritized in the *rc_buffer* for better system performance.

4. AIMM: ARTIFICIALLY INTELLIGENT MEMORY MAPPING IN NEAR-MEMORY PROCESSING SYSTEM

The resurgence of near-memory processing (NMP) with the advent of big data has shifted the computation paradigm from processor-centric to memory-centric computing. To meet the bandwidth and capacity demands of memory-centric computing, 3D memory has been adopted to form a scalable memory-cube network. Along with NMP and memory system development, the mapping for placing data and guiding computation in the memory-cube network has become crucial in driving the performance improvement in NMP. However, it is very challenging to design a universal optimal mapping for all applications due to unique application behavior and intractable decision space.

In this chapter, we propose an artificially intelligent memory mapping scheme, AIMM, that optimizes data placement and resource utilization through page and computation remapping. Our proposed technique involves continuously evaluating and learning the impact of mapping decisions on system performance for any application. AIMM uses a neural network to achieve a near-optimal mapping during execution, trained using a reinforcement learning algorithm that is known to be effective for exploring a vast design space. We also provide a detailed AIMM hardware design that can be adopted as a plugin module for various NMP systems. Our experimental evaluation shows that AIMM improves the baseline NMP performance in single and multiple program scenarios up to 55% and 50%, respectively.

4.1 Proposed Approach

We formulate the problem as a reinforcement learning problem as (1) we want a generalized and robust solution, which is applicable to all types of workloads, (2) the complexity and vastness of the problem space make this problem almost unapproachable with conventional deterministic algorithmic solutions, (3) technological advancements allow hardware implementation of the RL agent with very minimal cost and effort, which runs very fast and hence practical to be on the

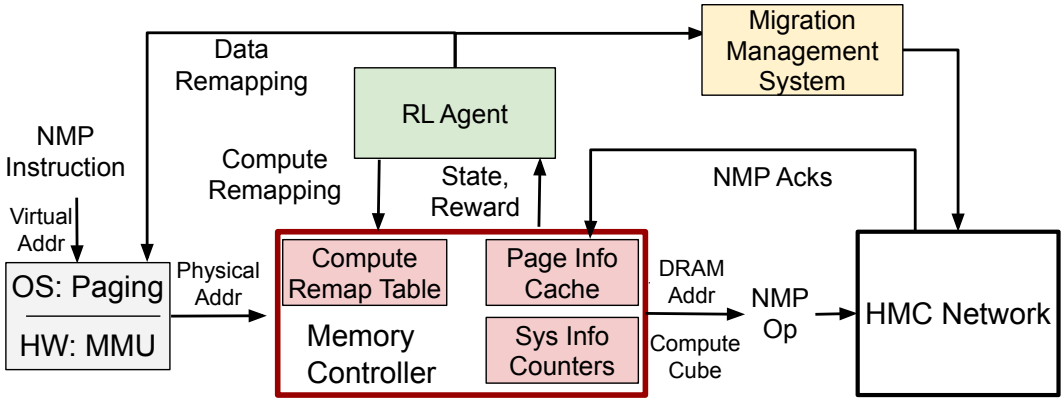


Figure 4.1: Overview of AIMM memory mapping for data and computation in NMP systems.

CMP chip. In this section, we introduce our proposed approach by first presenting the overview and problem formulation. Then we describe the representations of state, action and reward function for AIMM NMP memory mapping, followed by the DNN architecture and training of the RL-based agent.

4.1.1 Overview and Problem Formulation

AIMM continuously evaluates the data mapping done by the OS and computes cube decisions made by the memory controllers. Meanwhile, it provides suggestions to change not only the decision if necessary, but also the degree of the change (such as migration distance) in some cases. RL agent also learns the remapping and migration cost from the system feedback for each of its actions, and decides accordingly in the future (shown in Figure 4.1). The problem formulation is a challenging process due to the problem complexity and the lack of straightforward methodical ways. We explore two representations in this research: (1) Dataflow graph representation of accessed pages of an application; (2) Access history of each page along with the associated events (page miss, row-buffer hits, and vault accesses, etc.). The first one involves costly hardware resources for storing the dataflow graph and expensive Node2Vec embedding [166] for processing the states. So we chose the second one that is more cost-effective. We detail the representation of states, actions, and reward function of our formulation as follows.

4.1.1.1 State Representation

There are numerous parameters that can be considered for representing the state of the system holistically. In addition, the cost of information collection throughout the system may become prohibitively expensive. Hence, the state generalization and its representation are active research areas we continue working on. As an alternative, we narrow down the set of parameters, specific to our learning goal and represented them as it is. Our effort towards pre-processing the information, before using them to build the states needs further attention to make them cost-effective.

Based on domain knowledge, the state representation (ten parameters) consists of system parameters (four) and page access history (six), as summarized in Table 4.1. The RL agent takes this state information to improve the system performance through dynamic page-frame mapping. System information helps the RL agent to be aware of the system, while page information helps to focus on each page while taking an action. After collecting the required information of variable vector lengths, each of them is flattened into single vector of size 128×8 Bytes (128 64-bit entries). This leads to a vast space with $2^{128\times64} = 2^{8192}$ states in total in our system, making the problem very challenging.

System Information		Page Information	
Parameter Description	Vector Length	Parameter Description	Vector Length
NMP-op tab occupancy	n cubes	Page access rate	1
Row-buffer hit rate	n cubes	Migration per access	1
MC queue occupancy	m MCs	Hop count, Pkt lat	History length
Global actions	History length	Migration lat, Page action	History length

Table 4.1: The state representation of AIMM.

4.1.1.2 Action Representation

The actions are categorized into (1) computation migration, (2) page migration, and (3) training interval, as shown in Table 4.2. In addition, the default action (no change) allows the RL agent

to agree with the conventional system. "Near" and "Far" map to a randomly chosen neighboring HMC and the diagonally opposite remote HMC, respectively. Source co-location under page migration allows the system to co-locate NMP-op pages as a special case of page migration. The training interval regulates the epoch length in a range, discussed in §5.5.

Default	Computation Migration		Page Migration			Training Interval	
No change	Near	Far	Near	Far	Co-locate sources	Increase	Decrease

Table 4.2: List of actions sorted categorically.

RL actions need to be interfaced with both the agent as well as the actuator system. The neural network of the RL agent learns to decide the best action for a given input state through the feedback from the system provided in the form of reward. The actions can be interpreted by the system as it sees fit. However, system should stick to one interpretation of the actions for meaningful learning. In AIMM we assign and define the actions such that they are robust and flexible enough to be accommodated in any underlying system. Thus the interpretation of "near" and "far" is up to the system designer to take a decision on. We provide one of the possible interpretations and implementation in this dissertation.

4.1.1.3 Reward Function

As one of the most important components of the RL system, the reward function draws significant research attention [167, 168, 169] and demands a rigorous process (left for future work). We manually explore several reward functions, including use of magnitude of performance as the reward, which shows a slow or sometimes fluctuating learning curve. Hence, we digitize our reward function that returns a unit of positive (+1) and negative (-1) reward for performance improvement or degradation, respectively. Otherwise, a zero reward is returned. We have explored using the communication hop count as a performance metric, but it leads to a local minimum that does not reflect the performance goal. We empirically found that operations per cycle as a direct reflection

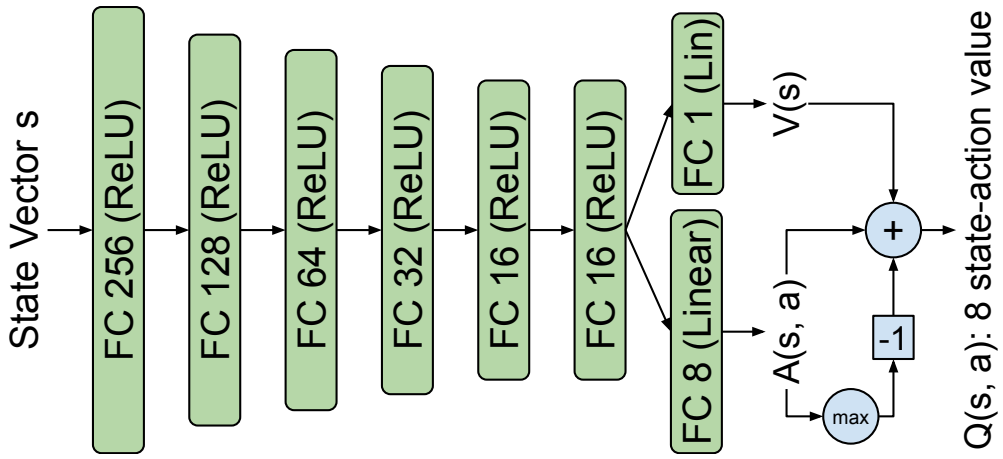


Figure 4.2: Dueling network for the RL-based agent. FC: Fully Connected, ReLU: Rectified Linear activation function. Total 8 action values, which are interpreted and realized by the underlying hardware once delivered.

of performance can achieve a robust learning process.

4.1.2 RL Agent

We use the off-policy, value-based deep Q-learning [121] algorithm for the proposed RL-based AIMM. For the state-action value estimation, we use a dueling network as a function approximator. The DNN model in the agent is a simple stack of fully connected layers, as shown in Figure 4.2. The agent takes the state and predicts the state-action value for each action. Then we use an ϵ-greedy Q-learning algorithm [120] to trade off the exploitation and exploration during the search and learning process. The algorithm selects an action randomly with probability ϵ to explore the decision space, and choose the action with the highest value with probability $1 - \epsilon$ to exploit the knowledge learned by the agent. To train the DNN, we leverage *experience replay* [121] by keeping the past experiences in the replay buffer and randomly draw the samples for training. Therefore, the learning and search process is more robust by consolidating the past experiences into the training process.

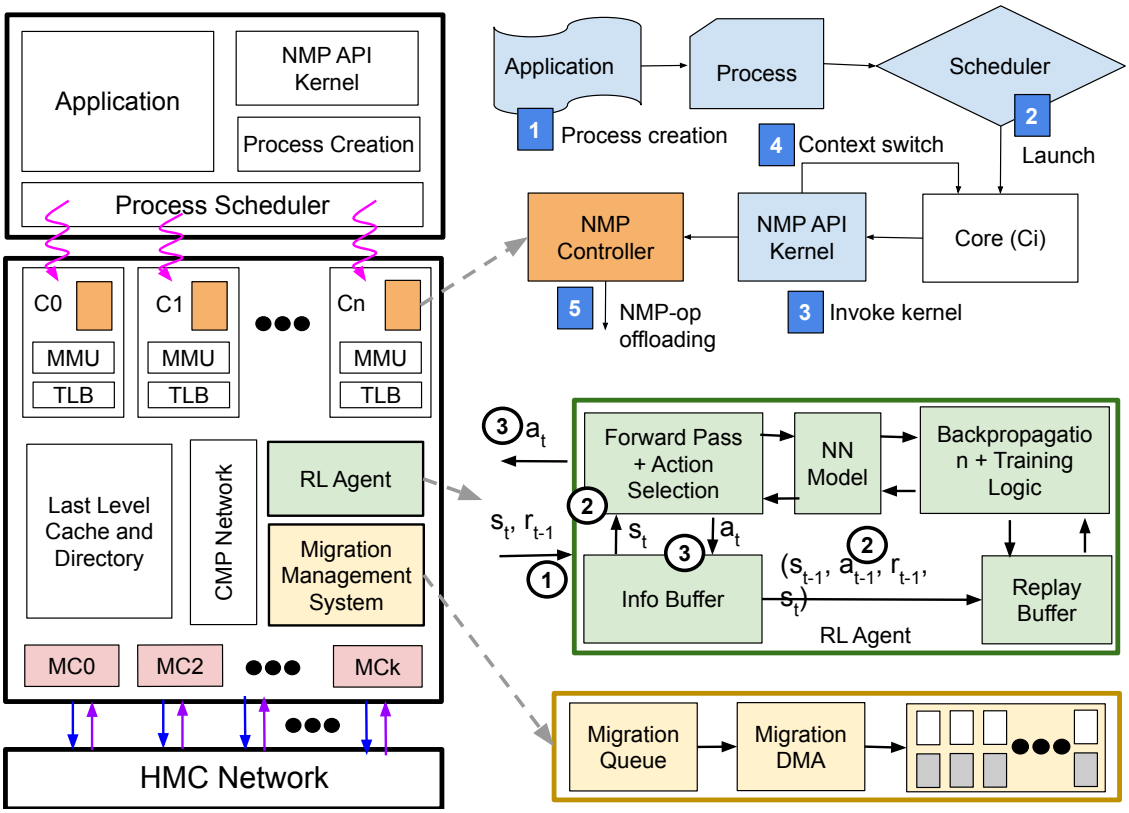


Figure 4.3: AIMM Architecture. In this diagram we show the flow of the AIMM system, where the whole system can be divided into three major cluster of components. The first one is the application and system software, where each application is registered as processes and on the system side the thread creation and thread scheduling is supported. The second one is the CMP core where NMP controller and RL agent is also situated along with other regular components. Finally, the HMC network connected with the MCs on the CMP and communicated through PCIe links.

4.2 Hardware Implementation

In this section, we detail hardware modules for implementing AIMM. As shown in Fig. 4.3, a CMP is connected to an HMC network (show in detail in Figure 4.4) through a set of memory controllers (MC) on the CMP side. The MCs on the CMP side (i) map the physical address to a specific HMC, (ii) generate HMC packets, and (iii) communicate with the RL agent. On the HMC side, each NMP-op is offloaded to an HMC as computation location based on the NMP technique, and corresponding memory operations are scheduled by the vault controllers as shown in Figure 4.4.

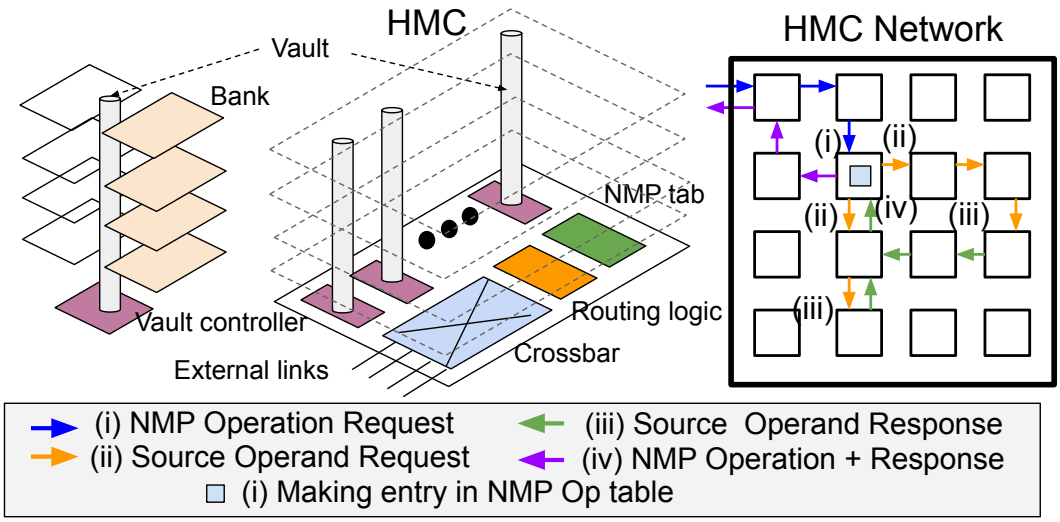


Figure 4.4: HMC network, where we show an example of basic NMP operation in the HMC network (where (i) NMP-op request enters through the corner HMC and makes entry in the NMP-op table, (ii) sends request packets, (iii) gets responses and (iv) computes in the NMP-op at the entry location), along with HMC dissection, featuring major HMC components.

4.2.1 Information Orchestration

Table 4.1 shows that the states are constituted using both system and page information, which are periodically collected at runtime. In MCs, the NMP-op occupancy is tracked for each HMC nodes stored in an array of registers, along with the MC queue occupancy information. Global actions are recorded by the RL agent itself. Row-buffer hit rate, however, is collected on the HMC node for each access and communicated to the MCs on the CMP side through the HMC network, in regular interval. Page specific information is stored in a fully associative cache like structure, using page number for identifying an entry. Each entry stores the frequency of page accesses (computed and stored on each access), frequency of migration for the selected page (incremented on each migrations), and history of hop count. Packet latency, migration latency, and actions taken for the selected page are also recorded. The history entries are temporarily stored in the HMC node using history-length number of registers and spilled into the MCs whenever the history registers get filled up or collection period/window ends. The cache uses the least frequently used replacement

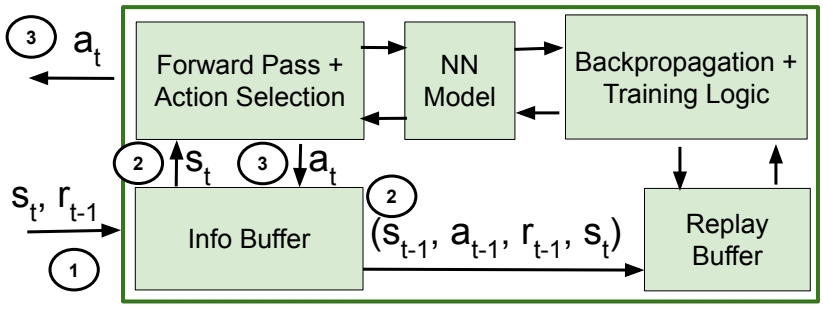


Figure 4.5: RL agent implementation through flow diagram.

policy. The state is constructed by combining all the information, and the reward is computed by comparing the current performance parameter value with the last computed value.

4.2.2 RL Agent Implementation

We propose to use an accelerator (as shown in Figure 4.5) for deep Q-learning technique following such accelerators that have been proposed in literature [170, 171]. The agent, running on the accelerator, pulls information from each memory controller. The incoming information, including the new state s_t and a reward r_{t-1} for the last state-action (s_{t-1}, a_{t-1}), are stored in the information buffer (①). The incoming information and the previous state and action in the information buffer form a sample $(s_{t-1}, a_{t-1}, r_{t-1}, s_t)$, which is stored in the replay buffer (②). Meanwhile, the agent infers an action a_t on the input state s_t (②). The generated action a_t is stored back to the information buffer and transmitted back to the MCs which then perform the appropriate operations (③). Upon the training time, the agent draws a set of samples from the replay buffer for training and applies a back propagation algorithm to update the DNN model.

4.2.3 Page and Computation Remapping

Page migration is managed by migration management unit and employed in the HMC network using migration DMA as shown in Figure 4.3. The TLB shootdown is customized to offset its latency overhead incurred when handled by the OS. Compute migration is implemented in the page granularity, meaning once the computation is migrated to a location, all the computation involving that page as destination (`dest`) will be done in that location.

4.2.3.1 Page Remapping

It involves OS for page table update and the memory network for page migration to reflect a page remapping, where the virtual page is mapped to a new physical frame belonging to a memory cube suggested by the agent. We provide *blocking* and *non-blocking* modes for pages with *read-write* and *read-only* permissions, respectively. For blocking migration, the page is locked during migration and no access is allowed in order to maintain coherence, which also experiences TLB shootdown overhead, implemented using constant latency. For non-blocking migration, the old page frame can be accessed during the migration to reduce performance overhead. In addition, the TLB is extended with an extra physical address field, which is used to hold the new frame number during migration [172]. When a page migration is requested by the data remapping decision from the agent, the page number and the new host cube are put into the migration queue of the migration management system. When the migration DMA can process a new request, the OS is consulted to provide a frame belonging to the new host cube and broadcast a message to update the extended slot for the new physical frame location in the TLBs. Then DMA starts generating migration requests that request data from the old frame and transfer it to the new frame in the new host cube. Once migration finishes, a migration acknowledgement is sent from the new host to the migration management system, which then reports the migration latency to the memory controller. Meanwhile, an interrupt is raised to invoke the OS for a page table update and notify the TLBs to use the new physical frame. In case of blocking migration, the page is unlocked for accessing, whereas for non-blocking migration, the old frame is put back to the free frame pool when the outstanding accesses finish.

4.2.3.2 Computation Remapping

Computation remapping decouples computation location and the data location for balancing load and improving throughput of the NMP memory network. Computation cube of an NMP-op is determined by the NMP-op scheduler based on the data address. The computation cube is then embedded in the offloaded NMP-op request packet. A compute remap table is used to remap the

computation to a different cube suggested by the agent. When a compute remapping decision related to a page is given by the agent, the page number and the suggestion are stored in the compute remap table. Upon scheduling an NMP-op, the NMP-op scheduler consults the compute remap table. If the related page of the NMP-op has an entry in the table, the computation cube is decided based on the agent suggestion recorded in the entry. Otherwise, the default scheduling is used. When the computation is done, the result is written back to the destination memory location.

4.3 Evaluation Methodology

In this section, we describe the simulation framework and methodology that implements and evaluates several NMP techniques and mapping schemes, respectively, followed by the workloads and analysis of their characteristics.

4.3.1 Simulation Framework

We develop a fast and accurate simulation framework (MC^{2}sim) to model RL agent and system architecture. The RL agent is functionally modeled using keras-rl [173] and built upon gym [174]. Since we propose to use a hardware accelerator for RL implementation (as discussed in §4.2.2), the timing aspect of the hardware accelerator is extracted using the MAESTRO inference model [175], considering training time as $2\times$ of the inference time. The simulation framework takes an event-driven approach to model a cycle-level system architecture with three group of components, namely, (1) Front-end (2) Chip-Multi Processor (CMP) and associated components, and (3) HMC Model and HMC network. The RL agent and the system architecture model are seamlessly integrated in order to achieve high simulation speed. The hardware configurations used in our evaluation are summarized in Table 4.3.

4.3.1.1 Front-end

The front-end of MC^{2}sim supports writing micro-kernels using a simple programming interface (equipped with overloaded memory allocation API, memory access API, etc). The trace manger facilitates real application traces and derives NMP operations. The processes that are registered during the initialization phase are launched on their allocated cores at their specified launching

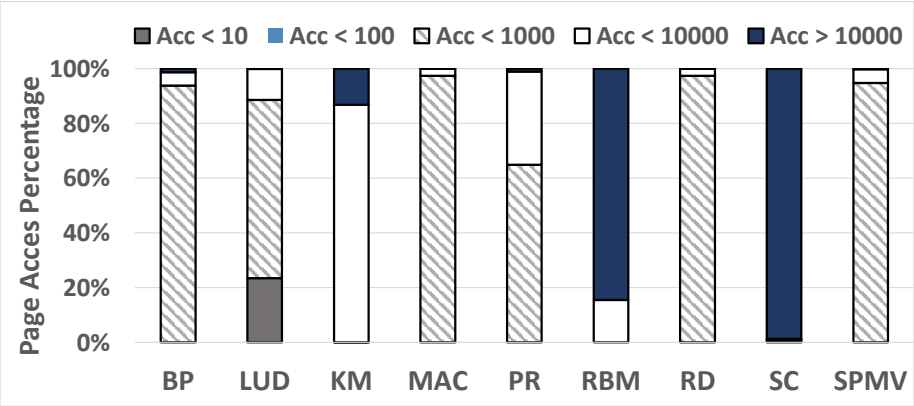


Figure 4.6: Classification of pages based on their access volume.

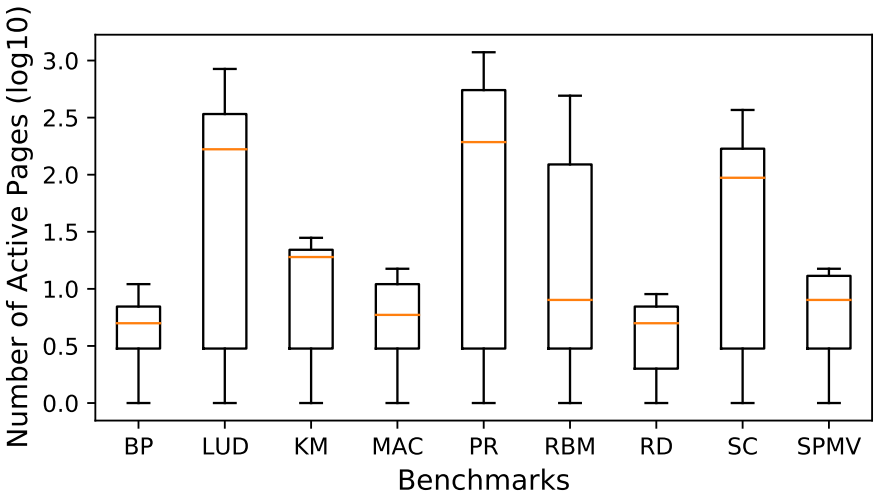


Figure 4.7: Page touched distributions representing number of pages touched in each epoch of 10000 cycles for the whole application execution

time by the process scheduler.

4.3.1.2 CMP and Associated Components

The event-driven implementation of CMP consists of a set of in-order cores, four Memory Controllers (MC), MMU units, and an NMP controller connected through a fixed latency high-bandwidth crossbar (to achieve high simulation speed). In a 64GB unified memory network formed using sixteen 4GB HMCs, MCs are responsible for determining the host cube of a given 36-bit

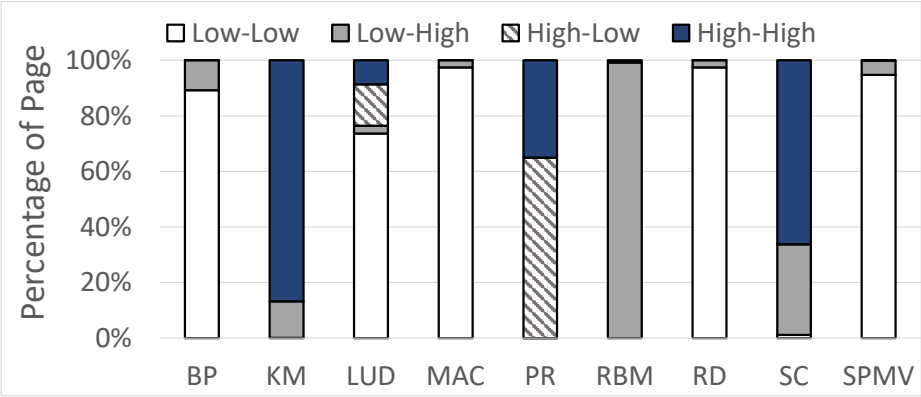


Figure 4.8: Page affinity showing the interrelation among the pages in an application.

Hardware	Configurations
Chip Multiprocessor (CMP)	16 core, Cahce (2MB/each core), MSHR (16 entries)
Memory Controller (MC)	4, one at each CMP corners, Page Info Cache (128 entries)
Memory Management Unit (MMU)	4-level page table
Migration Management System (MMS)	Migration Queue (128 entries), DMA (Rx,Tx buffers 128 entries)
Memory Cube	4GB, 32 vaults, 8 banks/vault, Crossbar
Memory Cube Network (MCN)	4×4/ 8×8 mesh, 3 stage router, 128 bit link bandwidth
NMP-Op table	512 entries
Hyper-parameters	reply-buffer (1000 entries), minibatch (128), learning rate (0.005)

Table 4.3: Hardware configurations.

physical address with 4 bits, and the remaining 32 bits are used to address (4GB) in each individual HMC. In addition, MCs create and send packets to respective HMCs, and also host page info caches to record page related information. The CMP, bus and the MCs have their respective queues, which backpressure NMP operation offloading; when they are full, forcing processors to stall fetching. The NMP controller helps create and offload the NMP operations to the HMC after the virtual-to-physical address translation, facilitated by the MMU unit equipped with a functional 4-level page table. The MMU also helps keep track of the migration candidates, and invokes DMA to initiate physical data migration through the memory network. Page faults are realized by employing process stall for a fixed number of cycles.

4.3.1.3 HMC Model and HMC network

The cycle accurate HMC network consists of event-driven HMC cubes [153] connected through the NI with the 3-stage pipelined router logic [130], and high bandwidth external SERDES links [176, 133]. The protocol deadlock is avoided using separate virtual channels (VCs) and network deadlock is avoided by using static XY routing. We aim for high-speed network simulation with simple round-robin allocation for the VCs and switch. The HMC reflects two different latency situations (row buffer hit and row buffer miss). Each of the banks has a row-buffer and each of the vaults has 2 banks connected in each layer (total 4 layers), constituting a total of 8 banks per vault. Parallel vault (32) accesses are facilitated by their respective vault controllers. NMP operations, are stored in an NMP-op table in each HMC, where each entry in the table connects to a simple Processing Unit (PU). The PUs receive operands either from the vault of the local cube, or from the remote cube through the crossbar that connects all the vaults and the NMP-op table with the external links.

4.3.2 Simulation Methodology

MC^{2}sim gives a parallel simulation view by running the hardware system and the RL agent in alternating epochs (100 to 250 cycles), where the RL agent recommends on the basis of system states in the last epoch to take action in the current epoch. The majority of the simulations are trace based, where the traces are collected by executing applications with data set-sizes ranging from 80 MB to 0.5 GB and annotating NMP-friendly regions of interest that were identified in previous works [18, 21]. The traces of an application form an episode for the application. AIMM evaluates both single-program and multi-program workloads, for several episodes (5 to 10), clearing the hardware system after each episode. The combinations of multi-program workloads are decided based on the workload analysis (§4.3.4). We evaluate the AIMM performance on the following techniques.

We create two variations of Active-Routing (AR) technique, based on the NMP-op computation point and named as (1) Basic NMP (BNMP) and (2) Load Balance NMP (LDB). The BNMP computation point is the HMC that contains the `dest` page of the requested NMP-op `<&dest`

`+= &src1 OP &src2>`. The LDB computation point is the `src1` operand location, owing to the observation that `src` operands span across more pages than `dest`, and hence are used for default load balancing. We use PEI unchanged as they send one operand data with the NMP-op request itself; the computation point is always the other operand location.

The TOM heuristic is adopted on top of the static computation point logic to improve performance through better resource management. It derives the node-id using a few bits from the physical address by using hash functions, decided after profiling for an epoch (10000 cycles) and applied in the following epoch. Multi-program scenarios make use of HOARD for localizing process data through page frame allocation using process specific free-frame list, in order to avoid inter-process interference. Unless otherwise mentioned BNMP is considered as our common baseline technique for result normalization and NMP operations are marked as non-cacheable [21]. We discuss each of the techniques in detail as follows.

4.3.3 NMP Techniques and Mapping Schemes

4.3.3.1 Basic NMP (BNMP)

We implement BNMP, following Active-Routing [21] for scheduling each operation in the memory cube while the in-network computing capability is not included. The NMP operation format is considered as `<&dest += &src1 OP &src2>`, where `&dest` page host cube is considered as the computation point. An entry is made in the NMP-Op table at the computation cube and requests are sent to other memory cubes if sources do not belong to the same cube as the destination operand page. Upon receiving responses for the sources, the computation takes place and the NMP-Op table entry is removed once the result is written to the memory read-write queue. The response is also sent back to the CPU if required.

4.3.3.2 Load Balancing NMP (LDB)

This is a simple extension of BNMP, based on two observations as follows. (1) Oftentimes, some NMP-Op table receives a disproportionate load based on the applications access pattern. (2) In most of the applications, the number of pages used for sources is significantly higher than

the number of pages used for destination operands. Hence we simply change computation points from destination to sources in order to balance the load on the NMP-Op table. However, once the computation is done, the partially computed result must be sent back to the destination memory cube and also to the CPU.

4.3.3.3 PIM Enabled Instruction (PEI)

This technique recognizes and tries to simultaneously exploit the benefit of cache memory as well as NMP. In case of a hit in the cache for one operand, PEI offloads operation with one source data to another source location in the main memory for computation. Being one of the early proposals on processing in-memory (PIM), they do not change the existing sequential programming approach where operation location is decided based on the locality of the data accesses. In terms of data and computation mapping PEI leaves fewer options open for AIMM, as they provide one of the operands along with the operation itself. AIMM can still explore computation migration and relative location of data from the connections with MCs, leading to some performance benefits as shown in Figure 4.9.

4.3.3.4 Transparent Offloading and Mapping (TOM)

This is a physical-to-DRAM address remapping technique, originally used for GPUs to co-locate the required data in the same memory cube for NMP. Before kernel offloading, TOM profiles a small fraction of the data and derives a mapping with best data co-location, which is used as the mapping scheme for that kernel. We imbibe the mapping aspect of TOM and make required adjustments to incorporate it in our context for remapping data in the NMP system. We infer data co-location from data being accessed by NMP-Op traces. Each mapping candidate is evaluated for a thousand cycles with their data co-location information recorded. Then the scheme with best data co-location that incurs the least data movement is used for an epoch.

4.3.3.5 HOARD

We adopted an NMP-aware HOARD [101] allocator as the baseline, heuristic-based OS solution for comparison and also as a foundation for experiment. HOARD is a classic multithreaded

page-frame allocator which has inspired many allocator implementations [177, 178, 179] in modern OSes. The original version of HOARD focuses on improving the temporal and spatial locality within a multi-threaded application. HOARD maintains a global free list of larger memory chunks which are then allocated for each thread to serve the memory requests of smaller page frames or objects. Once the thread has finished the usage of the memory space, it chooses to "hoard" the space in a thread-private free list until the space is reused by the same thread or the whole chunk gets freed to the global free list. As a result, HOARD is able to co-locate data that belongs to same thread as much as possible. We adopted the thread-based heuristic of HOARD for each program in our multi-program workload setting. Our HOARD allocator aims for improving the locality within each program, contributing to the physical proximity of data that is expected to be accessed together in the NMP system.

4.3.4 Workload analysis

We primarily target the long running applications with large memory residency, which repeatedly use their kernels to process and compute on huge numbers of inputs. The machine learning kernels are a natural fit for that as they are widely used to process humongous amounts of data flowing in social media websites, search engines, autonomous driving, online shopping outlets, etc. These kernels are also used in a wide variety of applications such as graph analytic and scientific applications. Since these are well known kernels, we describe them briefly in Table 4.4. To capture the page-frame mapping related traits of the application, we characterize the workloads in terms of (1) page usage as an indicator of page life-time, (2) number of pages actively used in an epoch as an indicator of space requirement in the page information cache, and (3) inter-relation among the pages (page affinity) accessed to compute NMP operations in order to analyze the difficulty level for optimizing their access latency.

4.3.4.1 Page Access Classification

The page usage is an indicator of its scope for learning the access pattern and improving performance after data or computation remapping at runtime. If the number of accesses to pages are

Benchmarks	Description	Input Data Size
Backprop (BP) [180]	Training feedforward neural networks.	2097152 hidden units
LUD [180]	Product of a lower and upper triangular matrix.	4096 matrix dimension
Kmeans (KM) [159]	Iterative algorithm, partitions the data-set into K pre-defined distinct non-overlapping clusters.	mfeat-zer [181]
MAC	Multiply-and-accumulate over two sequential vectors.	2×6400K dimension
Pagerank (PR) [13]	Rough estimation of the importance of the webpages.	web-Google graph [182]
RBM [12]	Variant of Boltzmann machines [183].	Variant of Netflix database [184]
Reduce (RD)	Sum reduction over a sequential vector.	6400K dimension.
SC [160]	Assigns each point of a stream to its nearest center.	10 20 128 16384 16384 1000
SPMV [159]	Solve sparse linear systems.	4096×4096 matrix

Table 4.4: List of benchmarks and corresponding input data sizes.

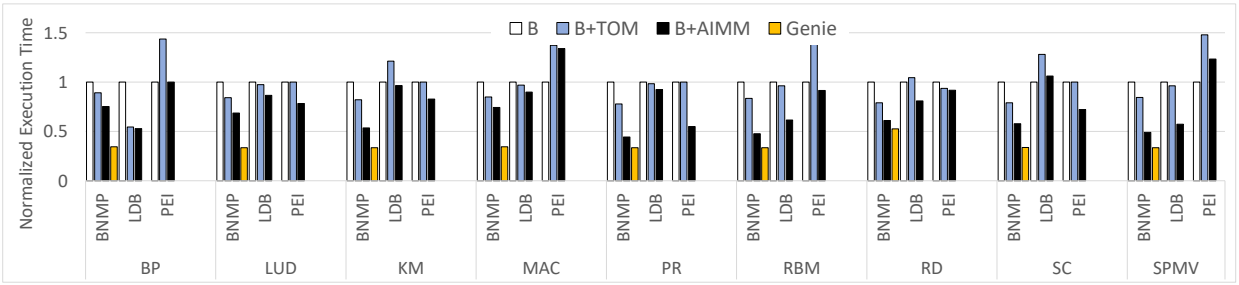


Figure 4.9: Execution time for all the benchmarks, normalized individually with their basic techniques BNMP, LDB, and PEI respectively, which does not have any remapping support, and commonly referred as B in the graph. Execution time for the baseline implementations are compared to the system with remapping support, namely, TOM and AIMM, respectively. We have added an unrealistic setup (Genie) result to show a highly optimistic upper-bound.

very low, the scope for improvement using remapping technique narrows down to a great extent. In Figure 4.6, we show that for most of our application kernels, the majority of the pages are moderate to heavily used over the whole application execution, which offers a substantial scope for AIMM.

4.3.4.2 Page Touched Distribution

Figure 4.7 shows the number of pages touched in an epoch of 10000 cycles. We observe low page reuse across epochs, leading to marginal benefits from using caches, which gets further amortized by high data movement cost. Hence, NMP offloading can save latency and bandwidth usage for these workloads. We can clearly identify two classes of applications. (1) High number of touched pages, *LUD, PR, RBM, SC*, and (2) Low or moderate number of touched pages, *BP,*

KM, MAC, RD, SPMV in the epochs. As opposed to the medium to large working set (80MB to 0.5GB) of these applications [21], this study gives us an indication of the amount of page information ideally needed to be stored in the page information cache in an epoch for the training of the RL agent.

4.3.4.3 Affinity Analysis

In data mining, affinity analysis uncovers the meaningful correlations between different entities according to their co-occurrence in a data set. In our context, we use and define the affinity analysis to reveal the relationship among different pages if they are being accessed for computing the same NMP operation. We track two distinct yet interlinked qualities of page access pattern, (1) the number of pages related with a particular page as the radix for that page, which is similar to radix of the node in a graph, (2) the number of times each pair of connected nodes are accessed as part of the same NMP operation, similar to the weight for an edge. To understand the affinity, we create N number of bins for each of the traits and place the pages in the intersection of both by considering the traits together. So the affinity space is $N(accesses) \times N(radix)$, which is further divided into four quadrants to produce a consolidated result in Figure 4.8. Based on our study, a higher affinity indicates a harder problem, which poses greater challenges for finding a near-optimal solution. On the contrary, they also exhibit and offer a higher degree of scope for improvement. Please note that this is only one aspect of the complexities of the problem. Interestingly in our collection, we observe a balanced distribution of workloads in terms of their page affinity.

4.4 Experimentation Results

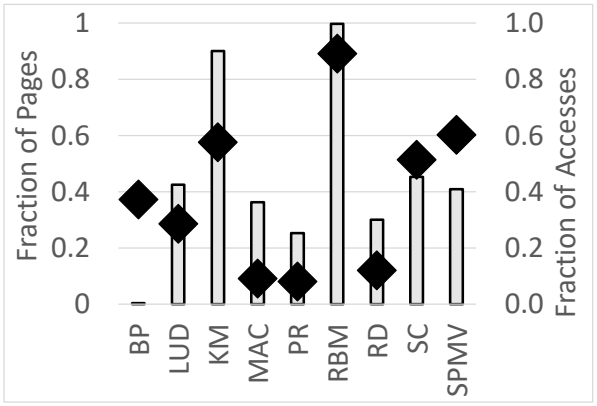
4.4.1 Performance

4.4.1.1 Execution Time

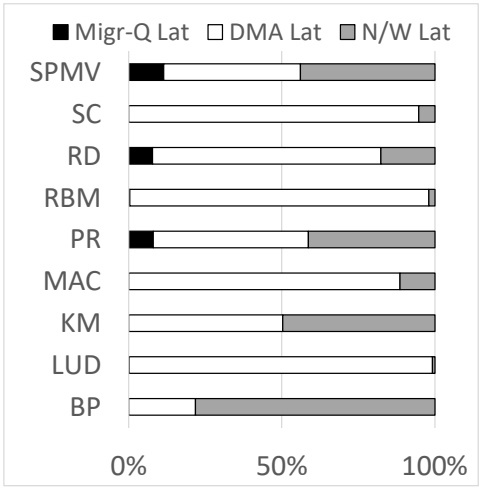
Figure 4.9, shows the execution time for different applications under various system setup and support. It is evident that AIMM is effective at helping the NMP techniques to achieve better performance in almost all cases. We observe up to 55% improvement in execution time. With BNMP processing, the NMP operations are completely on the memory side, both TOM and AIMM boost

the performance of BNMP. In general, TOM achieves around 15% to 20% performance improvement. AIMM secures improvement in execution time by around 50% on average across all the benchmarks. However for LDB, TOM did not improve much for most of the benchmarks. With LDB, AIMM improves execution time up to 43%, with no performance degradation observed except minor performance loss for *SC*. In the case of PEI, it is evident that for the majority of benchmarks TOM degrades the performance, whereas AIMM manages to achieve performance benefit around 10% to 20% (up to 42%) on average. For *SPMV* and *MAC*, both TOM and AIMM degrade the performance of PEI. There are several factors that play major roles in our system to drive system performance, such as operation per cycle, learning rate, utilization of migrated pages, path congestion and computation level parallelism. In addition, hop count in the memory network and row buffer hit rate in the memory cube can be considered as primary contributors for performance, depending on the nuances of individual cases. In the following, we discuss each of the techniques to justify their performance.

In the case of *PR* on BNMP, AIMM does not improve over TOM, which can mostly be attributed to the 3× computation distribution achieved by TOM over AIMM as shown on Figure 4.13a, in addition to very low migration page access for *PR* as shown in Figure 4.10. Figure 4.6 shows that *PR* has high number of pages that accessed small number of times, justifying low usage of migrated pages. Higher computation distribution improves NMP parallelism at the cost of extra communication if the computation node and operation destination are different. Low accesses to migrated pages diminishes the benefit of migration. On the other hand, AIMM achieves 50% better performance for *SPMV* that is ascribed to significant improvement in average hop count in Figure 4.13a, moderate fraction pages migrated (40%) and they constitute almost 60% of all the accesses as shown in Figure 4.10. The performance of *SC* with TOM and AIMM allows us to delve into discussion of the trade-off between computation distribution and hop count in the memory network, along with importance of distribution of the right candidate in the right place. Missing one of them may offset the benefit achieved by the other, as the case for *SC* with TOM. On the contrary, AIMM leads to a better performance than TOM with very moderate compute distribution and low



(a)



(b)

Figure 4.10: Migration Stats: (a) On the major axis we show the fraction of pages that are migrated for each of the applications using bars. On the minor axis it projects the fraction of total accesses that are happened on migrated pages, with diamond shaped markers. (b) Migration latency breakdown.

hop count as shown in Figure 4.13a. Since LDB and PEI both tend to co-locate the sources belonging to an NMP operation, they leave much lesser chance than BNMP type techniques for further performance improvement using remapping like TOM or AIMM. The performance improvement is mainly achieved by optimizing the location of the destination pages with respect to the source operand pages.

For estimating an upper bound of the execution time, we build a Genie (with unrealistic assumptions) that magically puts computation in the node with least congestion, and also instantly gets the operand pages in that node for immediate computations without any delay or overhead (not even memory access latency as shown in Figure 4.9, projects ≈12.5% standard deviation for improvement of Genie over AIMM, across applications.

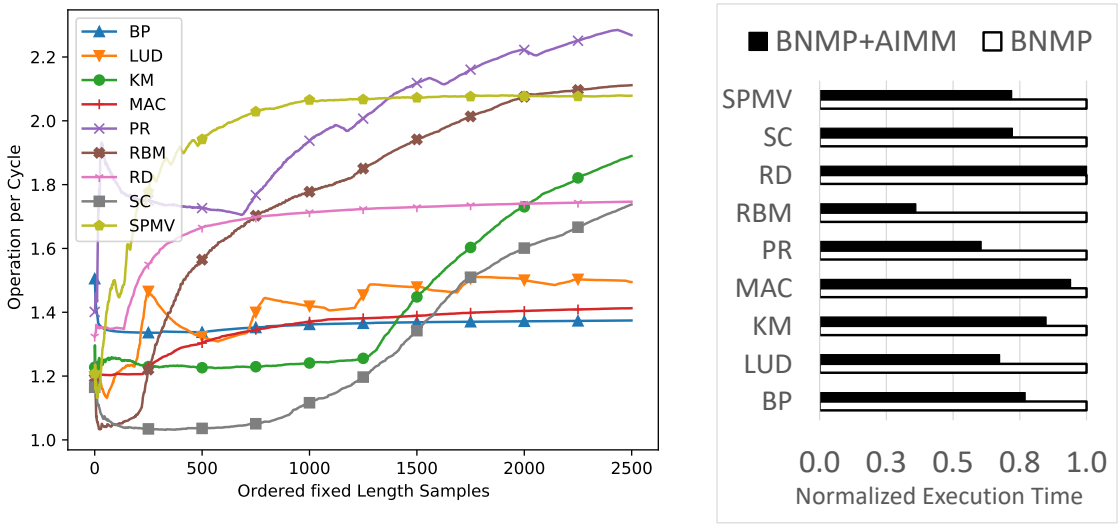
4.4.2 Learning Convergence

In Figure 4.11, we plot the timeline for OPC to show that AIMM progresses towards the goal for achieving high OPC as the time advances for each of the applications. Since the number of operations and so as the number of samples collected for each of the applications are different,

we randomly choose a fixed number of points while preserving their original order of creation. The agent learns that the page migration has a long-term impact on the benefit of data co-location, whereas the computation migration balances load by keeping the computation and communication resource availability in consideration. *KM* and *SC* still show potential to improve OPC further given more episodes to run and other applications converged.

4.4.3 Migration

In Figure 4.10a, we depict the fraction of pages being migrated for each application on the major Y-axis and the fraction of total accesses requested that belong to a migrated page on the minor Y-axis. The fraction of pages migrated is an indicator of the migration coverage. For instance, in the case of *RBM* 100% pages are migrated, and almost all the migrated pages get accessed later. Pages in *RBM* are susceptible to experience high volume of migration as (1) small number of pages are accessed most of the execution time, (2) in a small fixed time window almost all the pages are being accessed. On the other hand, *BP* has huge memory residency and relatively small working set, which leads to a low fraction of page migration as compared to the total number of pages. Interestingly, the small number of migrated pages constitute almost 40% of the total accesses, which is a near ideal scenario as low number of page migrations has a small negative impact on performance, and a high number of accesses to the migrated pages can potentially improve the performance, provided the decision accuracy is high. The RL agent should learn from the wrong migration decisions through the feedback, as they incur performance overhead. Figure 4.10b shows migration latency breakdown, where, depending on the application, the migration latency is distributed in different proportions between DMA and HMC network. We observe average migration costs $\approx$ 800 cycles to 2000 cycles, depending on the network congestion. That is why we use non-blocking migration for all of our experiments. The migration/access across all the applications ranges from 0.001% to 0.054%, which can be categorized as low to moderate frequency.



(a) (b)

Figure 4.11: (a) Operation per cycle timeline. The X-axis is the sampled time and the Y-axis is the value for OPC. The graph is not monotonically increasing as OPC depends on several system parameters at runtime. (b) Normalized execution time for 8×8 mesh (shorter is better).

4.4.4 Hop Count and Computation Utilization

In terms of system performance, hop count and computation distribution hold a reciprocal relation as reducing hop count improves the communication time, but results in computation underutilization due to load imbalance across cubes. On the other hand, computation distribution may result in a high degree of hop count as concerned pages can only be in their respective memory cubes. Hence, a good balance between these factors is the key to achieving a near-optimal solution. Figure 4.13a shows that AIMM maintains a balance between the hop count and computation utilization. For instance, *PR* and *SC* individually achieves very high computation utilization which also leads to high average hop count. Assuming that the computation distribution decisions and corresponding locations are correct, the benefit gets diminished to 22% and 20% respectively possibly because of high hop count.

4.4.5 Scalability Study

In this subsection, we extend our experiments to study the impact of AIMM under a different underlying hardware configuration (8×8 mesh) as well as under highly diverse workloads together

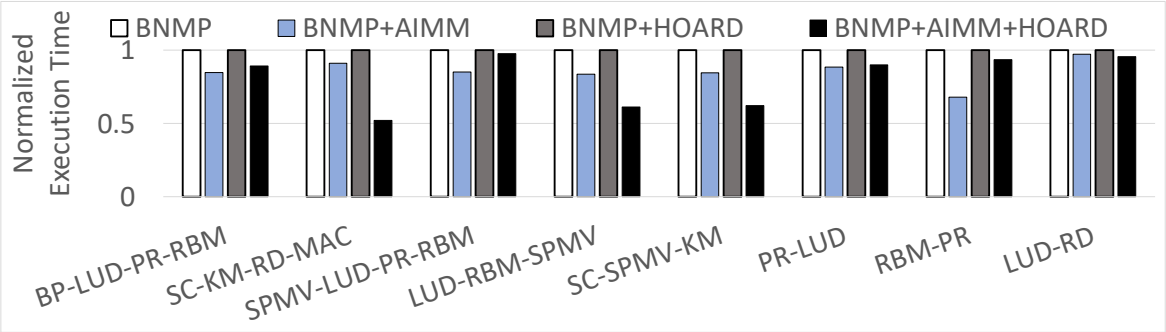
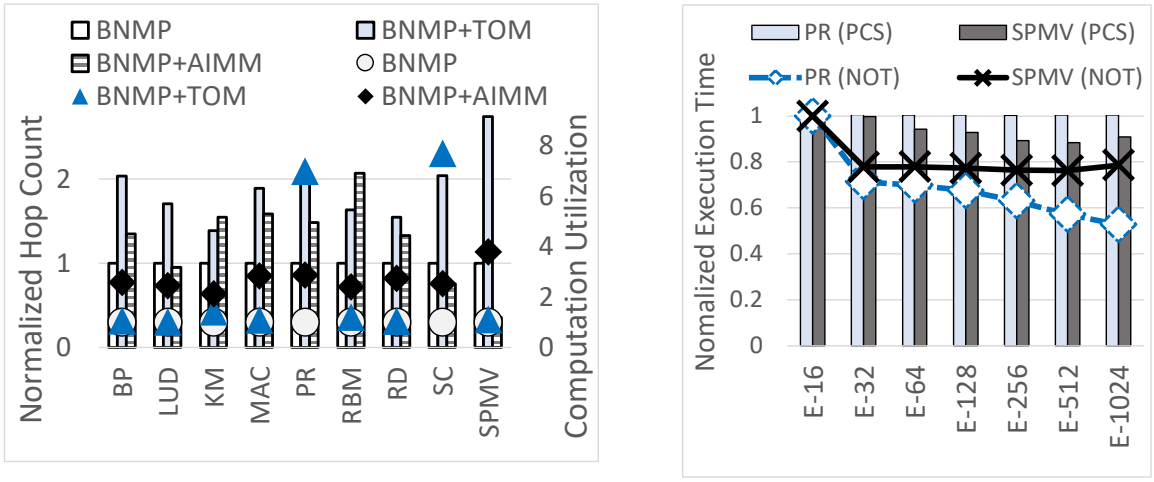


Figure 4.12: Multi-process normalized execution time. BNMP and BNMP + HOARD are considered as two separate baselines and used for normalize BNMP + AIMM and BNMP + AIMM + HOARD results, respectively.

(2/3/4-processes). With a larger network, we expect to observe higher network impact on the memory access latency, and intend to show that AIMM can adapt to the changes in the system without any prior training on them. While choosing applications for multi-program workloads, we select diverse applications together based on our workload analysis, so that the RL agent experiences significant variations while trying to train and infer with them.

4.4.5.1 MCN Scaling

We observe that AIMM can sustain the changes in the underlying hardware by continuously evaluating them and without having any prior information. However, the amount of improvements for the applications are different than that in a 4×4 mesh. For instance, as shown in Figure 4.11b, with BNMP+AIMM, *RBM* observes more benefit over BNMP, than it observes in the 4×4 mesh. As we know larger networks are susceptible to higher network latency, however, they also offer higher capacity and potential throughput. In the case of *RBM*, AIMM could sustain throughput, whereas benefits for other benchmarks slightly offset, mostly because of higher network delay. Note that in terms of simulation with larger network size, we did not change the workload size. Tuning hyper parameters is left for future work.



(a) (b)

Figure 4.13: (a) Average Hop Count and Computation Utilization. Major Y-axis is shown in bars. (b) Sensitivity study: The bar graph shows the sensitivity of the benchmarks for different page-cache sizes (PCS), whereas the line graph show the sensitivity to the NMP-Op table (NOT) sizes.

4.4.5.2 Multi-program Workload

Figure 4.12 shows multi-process execution time. For a continuous learning environment like AIMM, multi-program workloads pose a tremendous challenge on the learning agent as well as on the system. For studying the impact of multi-program workloads, we consider two baselines. (1) BNMP, where the NMP operation tables, page info cache, etc., are shared and contended among all the applications. We leave the study of priority based allocation or partitioning for future study. (2) BNMP+HOARD, where at the page frame allocator level our modified version of HOARD helps to co-locate data for each process, preventing data interleaving across processes. We observe that for several application combinations (*SC-KM-RD-MAC*, *LUD-RBM-SPMV*, *SC-SPMV-KM*), HOARD and AIMM compliment each other to achieve 55% to 60% performance benefits.

4.4.6 Sensitivity Study

We study the system performance by varying the sizes of (1) page info cache, whose size is critical for passing system information to the agent and (2) NMP operation table, whose size is important to hold the entries for NMP operations, denial of which affects memory network flow. We choose two representative applications (*PR*, *SPMV*) to study their performance by varying one

parameter while keeping the other one to its default value, mentioned in Table 4.3. (3) We also tune the training hyper-parameters for optimum results.

4.4.6.1 Page-info cache size (PCS)

Figure 4.13b shows that *PR* exhibits very minimal sensitivity to page cache size, whereas *SPMV* finds its sweet point while increasing the number of entry from 32 (E-32) to 64 (E-64). As a general trend, applications that get the most benefit from AIMM are more sensitive to the page cache size than others. Based on this study, we empirically decide the number of page cache entry as 256.

4.4.6.2 NMP table size (NMP-Op Tab)

NMP table size sensitivity depends on several parameters such as the number of touched pages (Figure 4.7), computation distribution, etc. In terms of average number of touched pages in an epoch, *SPMV* has around 10 pages on average in a time window. Figure 4.13a shows that *SPMV* has the highest computation distribution among the other applications. Combining these two pieces of data, it explains the reason for execution time saturation after 32 entries (E-32) for *SPMV*. On the other hand, *PR* has a very high demand for NMP operation table, as it has high average touched page count and low reuse rate. That is why *PR* shows monotonic improvement in execution time with the increase in the number of NMP table entries.

4.4.6.3 Training hyper-parameters

We study the sensitivity of the execution time with respect to replay-buffer size, frequency of training and learning rates. The replay-buffer size is varied $500\times$ observing only up to 4% performance loss, which helped to optimize energy and area consumption without hurting performance much. As discussed earlier, training rate is controlled by the agent by its actions (action #6 and #7), we vary the upper and lower bound of it and observe a sweet point (execution time ± 0.4 to ± 13). We apply the same strategy for learning rate as well, where we explore the spectrum to find the most suitable point (execution time ± 0.1 to ± 20) and set as default for all the applications and experiments.

4.4.7 Area and Energy

In this section, we discuss the detailed area and energy aspect of our design. The implementation of the RL engine and migration management unit demands a separate provision in the chip. For estimating the area and energy, we model all the buffers and caches using Cacti [185], 45nm technology, since these components contribute to most of the area and energy overhead in the system, and use McPAT [186] for overall CMP area estimation. Overall area overhead as compared to 16 core Xeon processor is ≈0.2% and HMC area overhead is even more negligible.

4.4.7.1 Information Orchestration

We estimated the area for hardware registers and page information cache as part of the information orchestration system. Since the hardware registers occupy negligible area, we mostly focus on the page information cache of size 64KB, which occupies 0.23 mm^2 area. The estimated per access energy for page information cache is 0.05nJ, which is consumed every time the cache is updated and read.

4.4.7.2 Migration

For the migration system, we consider three data storing points as a major contributor for area, namely, NMP buffer (512B, 0.14mm^2), Migration queue (2KB, 0.04mm^2), and DMA buffers (1KB, 0.124mm^2). It is worth noting that, depending on the organization and access method, the buffer and cache peripherals change significantly and so does their area. The per access energy consumption by these components are 0.122nJ, 0.02689nJ, 0.1062nJ, respectively. Page wise energy in the average case is composed of DMA leakage energy (15.51nJ) for waiting 500 cycles and the network cost for the whole page-frame transfer (574.44nJ), in total ≈ 600nJ.

4.4.7.3 RL Agent

For estimating the area and energy consumed by the RL agent, we focus on their major source of energy consumption that can easily be modeled as cache like structures, namely weight matrix (603KB, 2.095mm^2), replay buffer (≈512KB, 3.17mm^2), and state buffer (576B, 0.12mm^2). Their

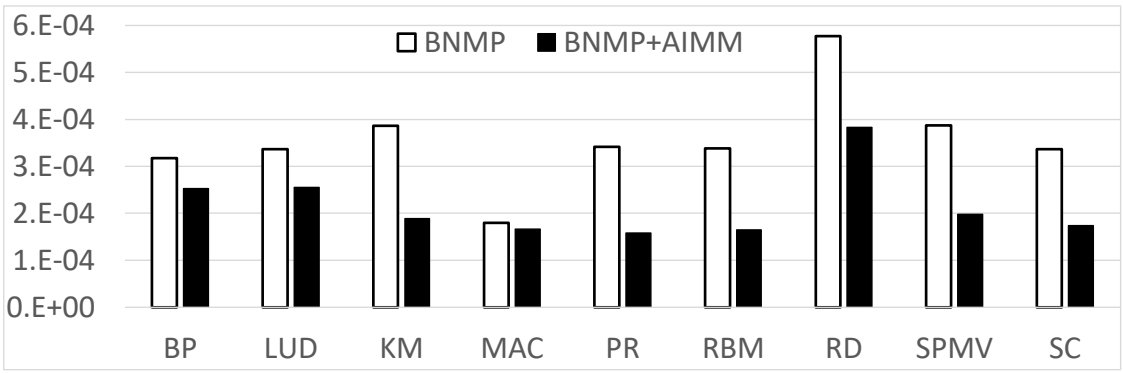


Figure 4.14: Energy-delay product (lower is better).

per access energy is estimated as 0.244nJ, 0.316nJ, and 0.106nJ, respectively.

4.4.7.4 Network and Memory

Since the migrations are realized by actually sending pages through the memory network with the help of DMA, we also estimate the network and memory cube energy consumption by assuming 5pJ/bit/hop [187] and 12pJ/bit/access [16] for the network and memory, respectively.

4.4.7.5 Overall Dynamic Energy

In our overall dynamic energy study, we include energy consumed by (1) only additional AIMM hardware, (2) memory network energy, and (3) memory cube energy, as major contributors to energy consumption in our framework. In Figure 4.14 we project Energy-Delay Product (EDP) as an overhead estimation metric, comparing the baseline and AIMM setup across the benchmarks. The EDP is roughly equivalent to the reciprocal of $MIPS^2$/Watt [188], where MIPS stands for Million Instructions per second and Watt (Joul/second) is the unit of power. In our case we replace instructions with NMP operations. The delay is the runtime, which is converted to seconds, assuming 2.66 GHz processor clock frequency. Overall EDP shows that, even though AIMM has some energy overhead over the baseline, it is still beneficial, as (except *MAC*) we observe that all the benchmarks' EDP is significantly lower than baseline.

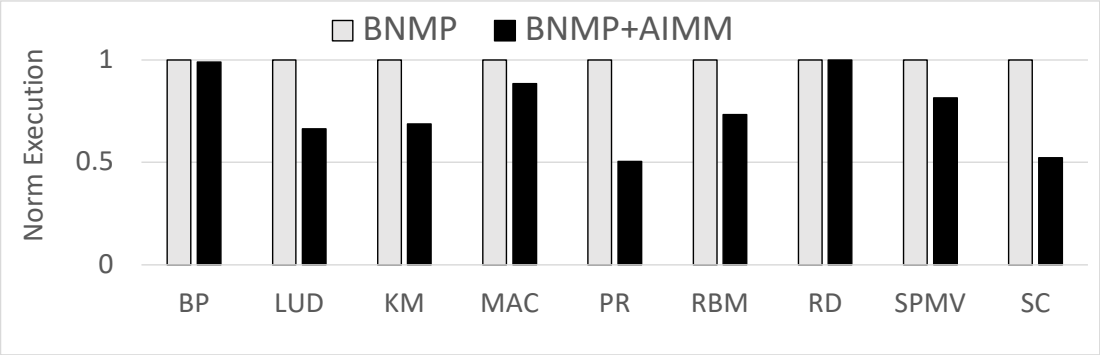


Figure 4.15: Normalized execution time for the HBM-PIM network. The X-axis is the benchmarks and the Y-axis is execution time, normalized to the BNMP.

4.5 Case Study: Scalable HBM-PIM

In this section, we discuss a case study and potential solution for addressing the capacity scalability issue with HBM-PIM [39]. It is pertinent to have a scalable solution, as many modern applications (good candidates for NMP) like, embedding lookup for embedding reduction [189], DNN kernel handling large input [190, 100], and other machine learning kernels used in our paper demands huge memory for handling large inputs. To evaluate the scope of AIMM in hierarchical HBM-PIM network, we develop a Transaction Level Model (TLM) [191], capturing high level functional behaviors of the hierarchical HBM-PIM.

To overcome the memory capacity scaling limitation along with computation bandwidth, we propose a hierarchical network of HBM-PIMs enabled with smart switches and connected through smart network interfaces (SMART-NIC). Considering current HBM-PIM as one tile, we connect 4 tiles (scaled up to 128GB) and a host CMP using a smart switch (5×5 ports, NMP computation enabled) with 100GB/s bus. The processor chip in the tile is used only for NMP computation and memory commands are scheduled by MCs, whereas the host CMP is used for operation offloading. The TLM model connects the modules using generic port, and uses generic payload and AXI4-Stream protocol [192] for communication; different constant latencies/buffer sizes are used to model memory access latency, computation delay, link accesses latency, reflecting their corresponding operating frequencies, bandwidth and pipeline behavior. We integrate this model

into MC^{2}sim. We provide occupancy for PEs (in switches/HBMs), read/write queues in HBMs, and generic ports as information to construct the states. Actions still dictate migrations (estimated overhead for page migration is less than 10% of execution time, is not considered) and the reward is constructed using OPC. The experimentation result is shown in the Figure 5.3. In our preliminary study we observe large variations in the improvement over baseline, ranging from none to 50% in execution time, possibly ascribed to optimization of computation location (external switch/tile switch/HBM-PIM) through improved data mapping. We observe that the $2\times$ improvement in execution time is partially attributed to better link utilization that happening for *PR* and *SC*, along with better resource utilization in terms of PEs. The benchmarks like *BP* and *RD* hardly gets benefit from AIMM as both observes a little scope for improvement in the hierarchical network. Other application kernels show moderate improvement in the results.

4.6 Summary

Careful articulation of data placement in the physical memory cube network (MCN) becomes imperative with the advent of Near-Memory Processing (NMP) in the big data era. In addition, scheduling computation for both resource utilization and data co-location in large-scale NMP systems is even more challenging than ever before. We propose AIMM, which is proven to be effective in assisting the existing NMP techniques mapped on MCN, by remapping their computation and data for improving resource utilization and optimizing communication overhead. Driven by the application's dynamic memory access behavior and the intractable size of data mapping decision space, AIMM uses Reinforcement Learning techniques as an approximate solution for optimization of the data and computation mapping problem. We project our technique as a plug-and-play module to be integrated with diverse NMP systems. The comprehensive experimentation shows significant performance improvement with up to 55% speedup for single-program workloads and up to 50% for multi-program workloads over baseline NMP. For broader application, AIMM can also facilitate data mapping in other near-data processing systems, such as processing in cache, memory, and storage.

5. MC^{2}sim: MEMORY-CENTRIC COMPUTATION SIMULATOR WITH PLUGGED-IN REINFORCEMENT LEARNING FRAMEWORK

The resurgence of near-memory processing (NMP) with the advent of big data has shifted the computation paradigm from processor-centric to memory-centric computing. Since memory-centric computing resembles more with our brain functionalities and has capability to exploit huge internal memory bandwidth and translate that into computation bandwidth, it may proven to be the future of commercial computation facilities. Researchers are working on several technologies like, DRAM chips (conventional DIMMs), DRAM cubes (HBM, HMC), NVM (ReRAM, PCM, etc.), which expands the application scope for memory centric computations in capacity extendable memory-network. However there is only a handful of free source simulation framework available to facilitate the memory-centric computation research.

Since simulation framework plays the key role in innovative architectural design, we delve into designing an end-to-end memory centric computation framework MC^{2}sim, capable of simulating Terabytes of in-memory data computations in reasonable timing. In addition, we also provide simple programming interface for writing simple kernel programs to mimic their memory access behavior. Owing to fact that there is a growing research interest in machine learning assisted computer hardware and policy design and decision making, we integrate a reinforcement learning agent which monitors the system parameters provided by the simulation framework and learns different traits of the system if enabled. The framework not only supports NMP operations, but also supports regular memory operations.

5.1 Overview of MC^{2}sim

In this section we describe the overall simulation methodology, design principle followed, and connection among the major class of components as shown in Figure 5.1. We can divide the whole framework in five different class of components ❶ kernel programming and real program trace support, ❷ chip multi-processor design, ❸ simple thread scheduling and page translation, ❹

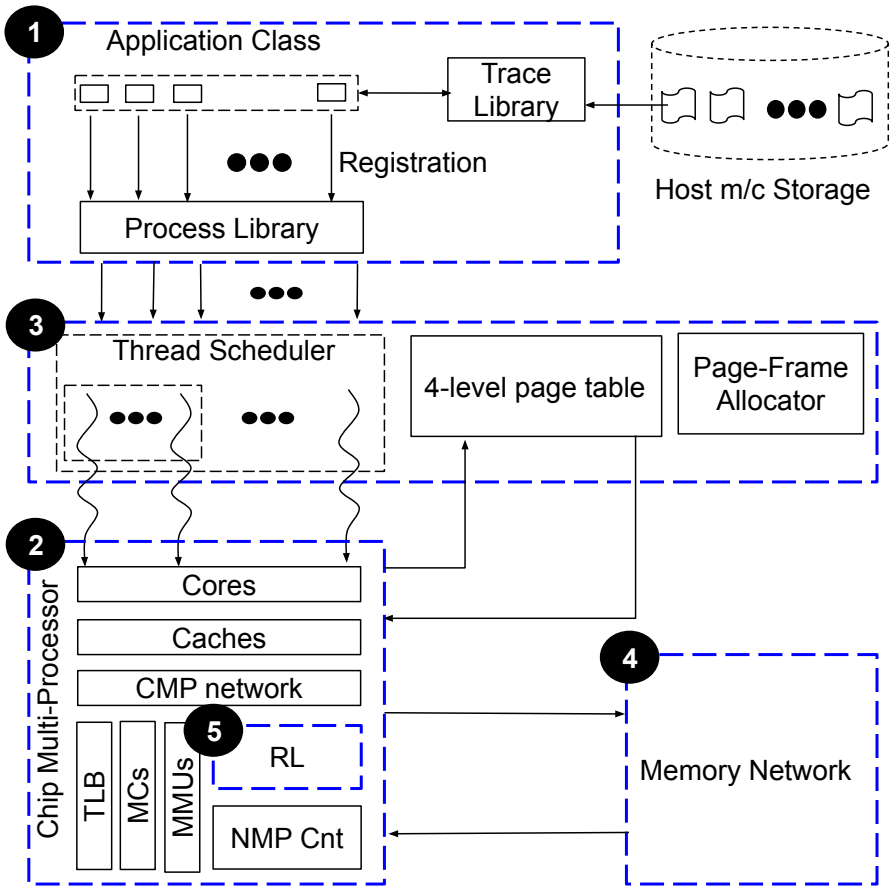


Figure 5.1: MC^{2}sim overview.

memory network, ❺ RL agent training and inference. We depict the way we put them together under the same memory-centric design framework. We briefly describe each of the classes and their integration process as follows.

5.1.1 Program kernel and Trace support

The simple programming interface allows user to write their own program in C++ and using programming APIs users can easily simulate the memory access pattern or NMP operations they want. The trace library on the other hand, allows users to create traces by running any application on the host machine and then run the real application traces on the simulator. The traces only contain virtual memory addresses, which are not related with the host machine. While simulating using these traces the physical configurations, virtual to physical address translation, etc., are

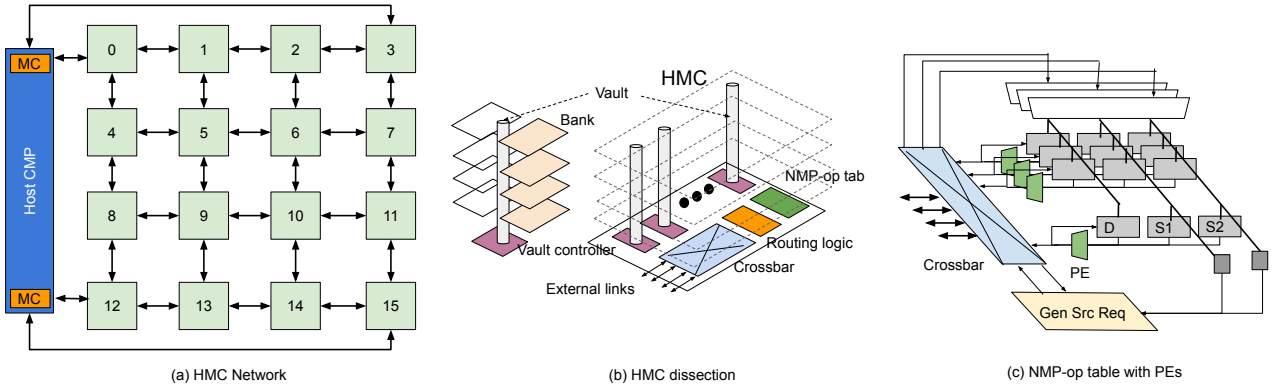


Figure 5.2: HMC network. (a) HMC network and connection with the CMP, (b) HMC dissection, (c) detail of NMP-op table, entries and connections.

applied and calculated according to the simulated system setup.

Both of these techniques together allow the users with a great flexibility of application choices. The operating system's process creation and handling is imitated using simulator's process library, which registers processes from a list of processes at run time. For each trace or kernel, there should be an interface written which registers the application trace or kernel as process to simulate at run-time. Each processes are pinned to different CMP cores and removed from that core only when the process completes. The number of applications that can be simulated in parallel is parameterized.

5.1.2 Chip multi-processor (CMP) Design

The CMP cores get the pinned processes at runtime through the thread scheduler, depending on the scheduling policy. At each cycle, CMP visits each and every core to check if there is an operation/instruction to schedule in that cycle. If an NMP operation or a regular memory request is ready to be scheduled, they are packetized and sent to the memory controller for further processing. The pipeline width of each core is parameterizable.

5.1.3 Partial OS support

Two OS kernels are supported so far for scheduling the threads in each core and also to support the address translation. On a TLB miss, the 4-level page table is invoked to get the address translation done. The translation is assumed to be done in constant 200 cycles, including the TLB update.

In the mean time the CMP core is stalled. Currently the thread scheduler only handles CMP core number of threads in the system and hence we do not simulate the process migration on context switch.

5.1.4 Memory Network

Once the translation is done for all the operands in an NMP operation, the NMP request packet reaches to the memory controller for finding out the actual physical location in the physical memory. Depending on the type of the memory technology, the job of the memory controllers may vary. In general if the memory is constituted of memory cubes, the memory controller decides the cube number in which the packet will be sent. Irrespective of the memory technology or network structure, each node in the network is equipped with routing and switching arbitration. The memory network implements the NMP operation process in the memory.

5.1.5 RL Agent

If enabled, the RL agent observes the memory system parameters and learns the way to reach the assigned learning goal. To allow the RL to learn, in the simulation framework it is invoked in a regular interval and carries out one iteration of training and one iteration of inference and a set of actions are delivered to the system for taking further actions to realize the suggestions.

5.2 Micro-architecture Modeling

In this section we describe the implementation by taking examples of HMC and HBM networks that MC^{2}sim supports by default, followed by brief CMP description, as MC^{2}sim aims to design a large memory centric computation system with in-build NMP support. We also show a detailed NMP functionality for both HMC and HBM-PIM network separately.

5.2.1 HMC Network

HMCs have in-built support for connecting each other using external links, which they call as chaining. HMC chaining allows a great flexibility of building large HMC network following any topology (router radix less than equal to four) as they forward memory requests to the correct

destination. In this section we first describe the HMC modeling followed by HMC enhancements for adapting NMP operations. Finally, we describe the HMC network design which by default fully supports NMP operation along with regular memory accesses.

5.2.1.1 HMC

As shown in Figure 5.2b, HMCs consist of four DRAM layers and one logic layer at the bottom. Each of these layers are divided into multiple banks. The layers are vertically connected using through silicon via (TSV) [193] and multiple banks (usually 2 to 4) in each layer are connected with it. The TSV latency and bandwidth is critical as they are responsible for (i) communicating memory commands between the vault-controller and the banks, (ii) responses from the banks to the vault controllers, and (iii) NMP-op results write back to the banks. At any point of time one TSV can carry on one of the above mentioned communications to and from a bank, connected to it. The number of TSV is parameterizable (usually 32 TSVs in an HMC). Each bank has a row associated with it and the HMC access latency is modeled by determining the row hit or row miss, considering TSV access latency inclusive. All the vaults are connected to the crossbar switch allowing communication among them. The crossbar has 32 internal ports and 4 external ports, altogether 32×36. This huge internal bandwidth has been supported and reflected in the model.

5.2.1.2 NMP-op support

To carry out NMP operation in the HMC, researchers propose to extend the HMC logic layer with array of simple computation units. In addition, for bookkeeping the operands an NMP-op table is also employed in the same logic layer. The maximum number of entries in the NMP-op table is parameterizable in the framework. Depending on the destination of the NMP operation, NMP-op table makes entry for that particular operation. Then checks the availability of the source operands. If they are in the same cube, where the destination is, they schedule regular read requests, putting the particular NMP-op entry in the NMP-op table as requester. Otherwise if the source operands are in different cubes, "Gen Src Req" block generates requests and sends them out through a particular port in the crossbar, based on the destination of the request following

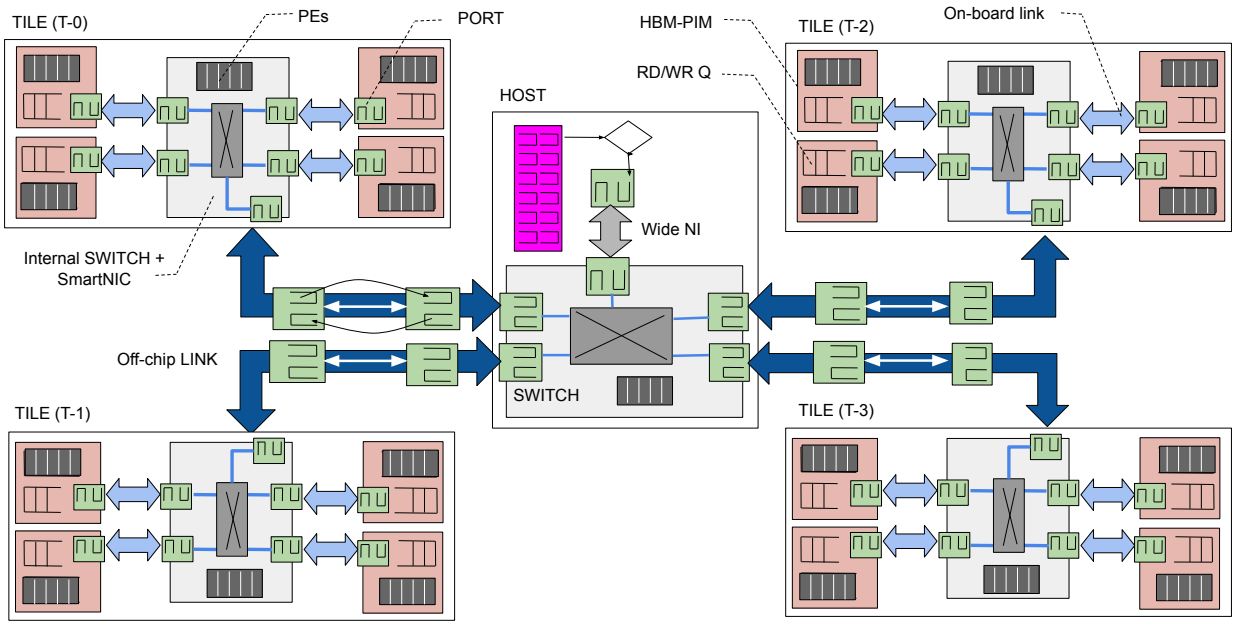


Figure 5.3: HBM-PIM network, the TLM model diagram. In the model each HBM-PIM cube consists of one read queue, one write queue (shown as one queue in the diagram) and a set of PEs. The set of PEs models the computations in the odd-even banks. Each of the HBM cube also has a generic port to connect with other devices, which also has the same generic port. Each tile consists of such four HBM-PIM cubes connected through a switch. The switch also has computation capability. The whole network is build by connecting four tiles through links with another switch, which also connects to one host that offloads the NMP-ops.

routing table.

Upon receiving a response for one or both the sources of an operation, the corresponding entry in the NMP-op table is being accessed. If all the operands are available, the computation is done by PE attached with that entry. The table entry is cleared once the result is forwarded to right location through the crossbar. Until then the computation result is stored in the destination (D) field in the same NMP-op table entry.

5.2.1.3 Building Network and associated protocols

The HMC network is built by connecting HMCs through SERDES [194, 195] links. The communication is done through generic payload. Payload is converted into several packets depending on the payload size and the link-width. We implement a credit based flow control, which incor-

porates the availability in the NMP-op table into account for the NMP-op requests that need to register in that particular node. The router is modeled as a three stage router and minimum amount of time a packet resides in the router is computed, based on the packet size.

In our model each HMC router has five ports, including the network interface. Each port supports up to six virtual channels (VCs), and number of buffers in each VC is parameterizable. We provide a mesh topology for HMC network and other topologies can also be built by providing corresponding network initialization files. However, for a different topology one must develop a supporting routing function. Currently, we support static XY routing for mesh. We model the link contention by allowing only one transaction at any point of time.

5.2.2 HBM-PIM Network

Even though both HMCs and HBMs are 3D stacked DRAMs, structurally HBMs are different from the HMCs. The major difference is that HBMs depend on other devices to implement memory controller, whereas HMCs implement controllers in its own cube and hence behave almost independently. In addition, unlike HMCs, HBM cubes do not provide routing and switching as well. Hence, forming HBM network is more challenging and involves some other supporting smart components for achieving high performance.

5.2.2.1 HBM-PIM

Recently proposed HBM-PIM [196] enables processing in memory (PIM) by implementing computations using the augmented memory circuit and reading values from the odd-even banks connected. In our model we assume that as long as the sources belong to the same cube, they can be rearranged to odd-even banks to carry out the PIM operation. The memory accesses are modeled by reflecting different constant latencies for row-buffer hit or miss. All the HBM requests are queued in read/write queues. The queue follows FIFO policy and number of requests serving per cycle is parameterizable. This is the first commercial prototype of HBM that supports PIM operations along with the regular memory accesses. In order to improve memory capacity, maximum four HBM-PIMs are connected with the CMP chip, beyond which extending memory capacity

faces difficulty because of physical pin limit in CMP chip. The CMP chip and four HBM-PIM cubes are connected through passive interposer links. Since passive interposer only supports direct links between two modules, increasing the number of HBM-PIM cubes exponentially increases the requirement for links in the interposer imposing another physical constraint on HBM-PIM capacity scaling.

5.2.2.2 Scalable HBM-PIM

As HBM-PIM work pointed out that there is a capacity limit of HBM-PIM as we already discussed, we propose a scalable solution, which involves and opens up opportunity in several different areas, like smart-switch, smart-NIC, etc., as shown in Figure 5.3. The detailed implementation of the smart components, we left for future work. We develop a Transaction Level Model (TLM) to implement a hierarchical network that consists of high bandwidth switches. The main switch is connected to CMP with a wide network interface (NI), which is capable of sustaining expected network bi-section bandwidth, $2\times$ of bandwidth of the individual links or switches. The switches follow round-robin scheduling for fair arbitration across input and output ports. The link contention is correctly modeled. We consider four HBM-PIMs, considering each of them as tiles, and connected the tiles with the main switch using generic ports. In fact, all the modules use the instances of the same generic port for creating network and facilitating communication. The mode of communication is generic payload which is recognizable in all the modules in the network. Each of the transactions follow simple AXI4-Stream in our HBM-PIM network model. The protocol can be changed easily if needed. The links that connect the main switch and the tiles supports upto 100 GB/s. The links in the HBM-PIM tiles can reach upto 350 GB/s bandwidth. The ports supports several virtual channels to avoid protocol level deadlocks.

5.2.3 CMP design

In Figure 5.4 we show the CMP design in MC^{2}sim. There are "k" processors which are capable of processing NMP kernels and "k" is parameterizable. The applications that run on these processors create processes to pin on each of the processor cores. Each of the processor in the CMP

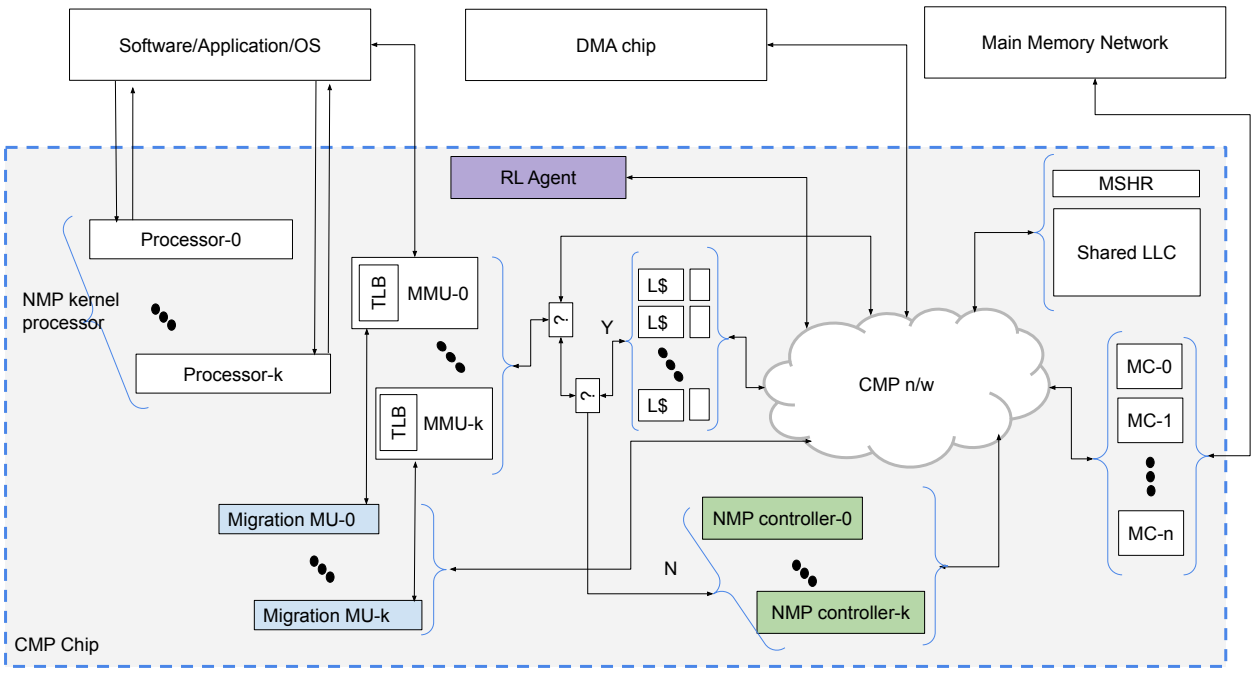


Figure 5.4: The CMP design.

belong to different CMP nodes that are connected through the CMP network. Each CMP node consists of a processor, memory management unit (MMU), migration management unit, NMP controller. Each processor has its private L1 cache. Each of the MMU handles the TLB, which is also private to each node. In addition, CMP has RL agent, shared last level cache (LLC), memory controllers that are common to all the CMP nodes.

The pinned processes offload NMP operations on each cycle and the MMU translates the virtual addresses with the help of TLB. In case of TLB miss, MMUs access the page table situated in the main memory. Once the translation is done for all the operands in one operation, the operation is sent to the NMP controller, which creates NMP operation for the NMP units in the main-memory and send them to the MC(s). The NMP operations are usually considered as non-cacheable and hence after address translation the NMP controllers are invoked. Other regular memory requests go through cache memory and reaches to MCs. In the MC(s) the NMP request and regular memory requests are handled equally and based on the underlying memory technology MC takes the

required course of actions. Since we are advocating for memory centric design, the memory network is quipped with its own memory controllers for handling the memory commands as described earlier.

To facilitate learning, we integrate an RL agent in the CMP and feed system parameters through the CMP network. To keep the simulation speed in the acceptable range we execute the RL agent on the host machine directly. However, to obtain a realistic simulation framework we estimate the average time for one training iteration and switch the execution control between system simulation and RL accordingly to provide a parallel simulation view. We also provide hardware accelerated page migration unit as an example to reflect on the RL actions at the runtime. There could be several other ways to reflect the learning outcome in the system, which needs to extended further.

5.3 Functional Component Models

To maintain the simulation speed we refrain from timing simulation for the learning accelerator and some part of the OS kernels. These components provide the the accurate functionality running directly on the host machine on which the framework is also running. In addition these components are also compiled in the same framework build so that they can be accessed within the framework process, for saving further inter-process communication delay during simulation.

5.3.1 RL components

We use the off-policy, value-based deep Q-learning [121] algorithm for the proposed RL-based AIMM. For the state-action value estimation, we use a dueling network as a function approximator. The DNN model in the agent is a simple stack of fully connected layers. The agent takes the state and predicts the state-action value for each action. Then we use an ϵ-greedy Q-learning algorithm [120] to trade off the exploitation and exploration during the search and learning process. The algorithm selects an action randomly with probability ϵ to explore the decision space, and choose the action with the highest value with probability $1 - \epsilon$ to exploit the knowledge learned by the agent. To train the DNN, we leverage *experience replay* [121] by keeping the past experiences in the replay buffer and randomly draw the samples for training. Therefore, the learning and search

process is more robust by consolidating the past experiences into the training process.

5.3.2 System components

Since the system components are same across all the system configurations, keeping the simulation speed in mind we functionally implement the following three OS kernels, (1) thread scheduler, (2) page table, and (3) page-frame allocator.

5.3.2.1 Thread Scheduler

The thread scheduler accepts all the process threads registered in the system and checks if system simulation reached the scheduling cycle for that thread. If the time for scheduling is reached, the thread is pinned to a CMP core till the end of the process and removed from the core once done. MC^{2}sim supports core number of processes to be scheduled to keep the thread scheduling simple. However, if some research needs to have more complex thread scheduler, it can easily be extended.

5.3.2.2 4-level Page Table

The four-level page table is built for accurate virtual to physical address translation and also account for the page faults in the simulation. The physical address is assumed to be 48 bits and the virtual addresses are considered to be 64 bits. The memory consumption of the page table could be unbearable if the whole page table structure needed to be allocated in the beginning of the simulation. We create the entries in the page table on demand, keeping the memory requirement for the page table as minimum as possible. The 4-level page table consists of three layers of directories and the penultimate layer with actual page tables in a complete graph tree structure. However, practically only a few branches of the tree is only required to be created for running most of regular applications.

5.3.2.3 Page-frame Allocator

We adopted an NMP-aware HOARD [101] allocator as the baseline, heuristic-based OS solution for comparison and also as a foundation for experiment. HOARD is a classic multithreaded

page-frame allocator which has inspired many allocator implementations [177, 178, 179] in modern OSes. The original version of HOARD focuses on improving the temporal and spatial locality within a multi-threaded application. HOARD maintains a global free list of larger memory chunks which are then allocated for each thread to serve the memory requests of smaller page frames or objects. Once the thread has finished the usage of the memory space, it chooses to "hoard" the space in a thread-private free list until the space is reused by the same thread or the whole chunk gets freed to the global free list. As a result, HOARD is able to co-locate data that belongs to same thread as much as possible. We adapted the thread-based heuristic of HOARD for each program in our multi-program workload setting. Our HOARD allocator aims for improving the locality within each program, contributing to the physical proximity of data that is expected to be accessed together in the NMP system.

5.4 Supporting Simulator Components

The four main supporting aspects of any simulator is the (1) reading inputs, (2) reading configurations, (3) output the simulation statistics, and (4) debugging. Since we support trace based simulation, MC^{2}sim facilitates reading trace files one by one, given trace location as the input to the simulation. It also supports simulation specific configurations and uses them to set parameter values either to configure the simulator, or setting some required simulation flags. The statistics for many parameters are delivered in a file at the end of the simulation. We also provide debugging facility for functional correctness of the implementation and logging.

5.4.1 Trace Support

The trace reader reads one trace file at a time for an application and store the whole content in a vector of trace elements. Each of the trace element stores one entry on the trace file. During the simulation, the processor fetches operations from the trace element vector and moves forward for further processing. When back-pressured, the operation fetch is halted to reflect the idle time. Once the content of the whole trace file simulated, the trace reader reads the next trace file. This is how it maintains the order of actual execution of operations done during creating the traces.

5.4.2 Setting and Reading configurations

MC^{2}sim facilitates the configuration reading as text, following the configuration format. Adding a new configuration is as easy as registering the new parameter in the configuration map that belongs to the configuration class, and then access through configuration objects in the places where needed. There should be one configuration file for each simulation, which is used to reconfigure some parameter that is need to be different than default. We support a wide variety of scalar parameter types like boolean, int, float, string, and vector as well.

5.4.3 Stats Collection

Adding and printing stats is the final goal for any simulation framework. We provide a stats parameter registration method and allow the description of the parameter during registration. The added parameter will automatically appear in the statistics output file along with its descriptions. The only thing the user need to set is the value that stats parameter should be updated with at the end of the simulation.

5.4.4 Debugging support

MC^{2}sim can be debugged with "gdb" and "valgrind" directly if compiled with the debug option. In addition, for functional verification we have extensive debug messages which can be enabled whenever required. We support total four classes of debug messages, each correspond to each class of components in the simulator as described earlier. The output of the debug messages are automatically written in the "debug.log" file. In addition we also provide a way to trace back to the network deadlock, (1) first by finding a particular packet stuck in deadlock and (2) then printing debug messages only for that packet. In addition, in case of a possible network deadlock situation at runtime, all the network buffer contents, mostly packets' source and destinations are printed in a human readable format for finding out the deadlock formation cycle.

5.5 Simulation Methodology

The main contribution of this framework is to decouple the learning agent execution and system simulation, yet run them together as part of the same process. In reality we want to run the system and learning agent in parallel. One way to achieve that in the simulation is to create two different threads, one for the learning agent and the other for the system simulation for executing them in true parallel mode. The threads can be synchronised based on the cycle count in the system simulation and one iteration of the training. The other way is to execute them in time division multiplexing fashion, where the same synchronization condition can be applied for switching the execution control. Since the later one is simpler to handle, we have opted that for the framework. The framework is also capable of running multi-program workload and as of now it is not equipped to execute multi-threaded applications.

The framework can take applications as input in two different ways, (1) trace, (2) c++ programs written using programming APIs provided by MC^{2}sim. The traces allow MC^{2}sim to execute any program irrespective of their source types and host requirements for execution. In addition, the simulations are very fast and focused as the programs need not to execute repeatedly once the traces are created. The only downside is the storage requirement forces us to trade-off the system details recorded in the traces. On the other hand, the programs written using the given APIs, sometimes experience difficulty to write the generalized conventional program itself as they are tightly integrated with the system simulation. Hence, we have only implemented application kernels which uses NMP by utilizing the second method.

5.6 Validation and Experimentation

In this section we try to establish the fact that MC^{2}sim possesses most of the traits that are essential for any simulation framework. First of all the functionality of the simulator must match with its intention. Based on the goal of the simulator, the level of accuracy should be set and achieved. Many simulators provide several options to the users to choose the level of accuracy they want. The simulation stability both in terms of result reproducibility and simulation software reliability are

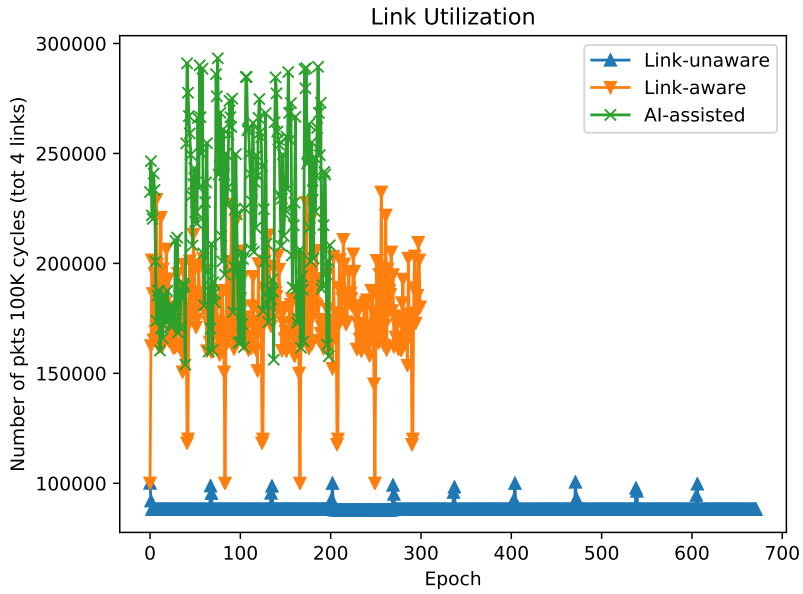


Figure 5.5: Link utilization of different policies on HBM-PIM network. The X-axis is the epoch of length 100K cycles. The Y-axis is the cumulative number of packets transferred through all the four links in one epoch. Higher Y-axis value shows better throughput.

essential qualities for a simulator. Above all simulation speed indirectly directs research throughput by having short simulations, allowing to try several options for a particular implementation and also help to achieve result coverage for wide range of configurations for research completeness.

5.6.1 Functional Correctness

Functional correctness in the context of MC^{2}sim needs to ensure that (1) all the NMP-ops are committed, (2) at the end of the simulation the network if properly drained, (3) NMP-op implementation being very complex in terms of its execution, it is very deadlock prone and hence, the implementation must the guaranteed to be deadlock free, (4) the memory row-buffer hit or miss should be accurate, (5) page-hit or page-miss should be implemented correctly in the page-table implementation and accurately reflect its latency in the simulation. We carefully checked all these functionalities by putting checking criteria in the simulation itself and by running several target specific verification codes. The checks and assertions will also trigger in case user implements something new which does not match with the original assumptions or the new implementation is

doing something wrong. For instance, a simple check if all the issued NMP-ops are got committed and some of them are not just lost; for avoiding protocol deadlock we provide several virtual networks and for network deadlock we provide credit based flow-control, etc.

5.6.2 Results Accuracy

MC^{2}sim is not intend to provide end-to-end execution time as absolute numbers, rather compare across different NMP techniques in the memory-centric framework, simulating the processor side in an abstract way for gaining high simulation speed. We borrow the idea of abstraction from processor centric simulators, most of which either implement different aspects of memory or network-on-chip in a high level or sometimes do not even implement it. Since we are designing memory-centric design we choose to abstract the CMP and hence we don't claim the end to end timing in absolute sense. However, since most of the important aspects of different NMP techniques are implemented in details, the comparisons across the techniques are fair. There will always be more for more perfection.

In any network system, link utilization is a good indicator of the network performance. In Figure 5.5 we show the link utilization (HBM-PIM network) for three different techniques which are (1) link-unaware, (2) link-aware and (3) AI-assisted (further details are out of the scope of this discussion). We want to show that given a better technique our link bandwidth can sustain higher traffic. It also shows that our AI-assisted technique successfully learns the traits of the link and improves its bandwidth over a period of time. Even though link utilization alone does not guarantee a better performance, but network based system should have a bandwidth elasticity. The peak bandwdith achieved is around 269 GB/s in total for four links, similar to regular PCIe bandwidth, around 100 to 120 GB/s. The Y-axis of the graph shows number of packets in 100K cycles, which derives to 2.7 packets in one cycle as peak bandwidth across four external links. Considering packet size 128 bytes, we get bandwidth per cycle $\approx$345 Bytes/cycle. Considering clock of 800 MHz (usual for PCIe bus) we get the bandwidth. The default link configurations can reach upto 100 GB/s for each link and can be configured for any bandwidth required by changing (i) the packet processing delay, (ii) number of packets processed per cycle, and (iii) the packet

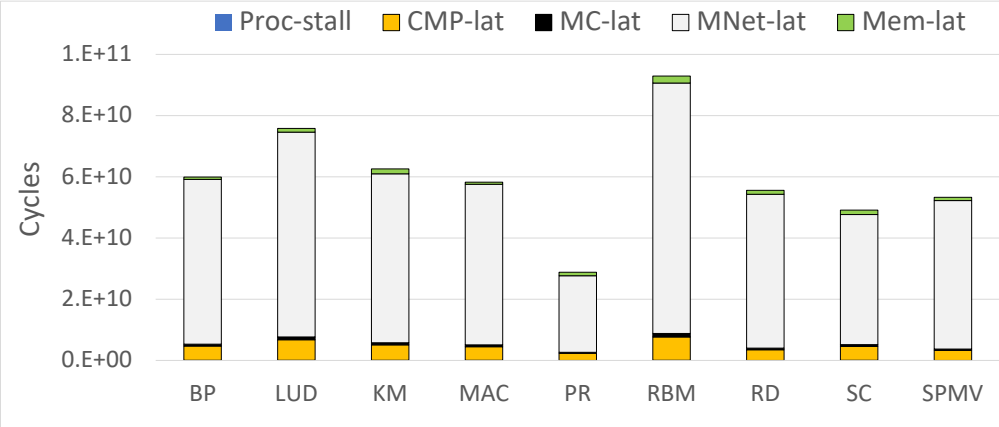


Figure 5.6: Operation Breakdown, collected by accumulating the latency experienced by each NMP-op in each stage. Proc-stall indicates number of cycles the process being stall for some reason, CMP-lat shows the latency in the CMP network, MC-lat shows the latency the NMP-op experienced in the MC on the CMP side, MNet-lat show the latency NMP-op experienced in the memory network, Mem-lat is purely memory access latency.

buffer size.

In Figure 5.6 shows the breakdown of the NMP-op latencies for all the NMP-ops in the application as a cumulative latency for each stages of the operation in HMC network. It is evident that the majority of the latency is contributed by the memory network because of the complex nature of the operations of the NMP-op as discussed earlier. The memory network latency includes all the latency for the NMP-op staring from reaching the memory network till it is added in the memory write queue after the result calculation. Depending on the policy, an NMP-op may result into five-to-six request/response communications in the memory network. It is expected to have a lower latency in the CMP network as we are trying to simulate a high bandwidth system, where the CMP is only act as a means of NMP-op offload.

5.6.3 Stability

The simulation stability can be defined in many ways. Here we define stability in terms of result reproducibility, simulation reliability and compilation support. Given an application and a set of hardware configurations, the simulator should produce exactly the same set of statistics values every time it simulates. Not only that, any change in the configuration should also be reflected in

some way if exercised by the application. In MC^{2}sim we make sure that above mentioned both cases are satisfied. Like any software, simulators also should be bug free and should be very reliable as long as the host machine satisfies the resource requirements. For instance, how much main memory is required to run the simulations. Moreover, the simulation should not crash suddenly because of some unseen bug in the code. In terms of compilation, if should support different compilation options including different compilers and their optimization options. We have tested our code with gcc-4.8 and gcc-8 for "-O0", "-O2" and "-O3" flags successfully. However, we need to have more constrained versions of python-3.6 and corresponding libraries like tensorflow (==1.13.1), keras, gym (==0.17.1).

5.6.4 Simulation Speed

Simulation speed being one of our desired design goal, we tried to come up with several implementation choices to make it as fast as possible, without compromising the accuracy and stability. It can simulate least 15K cycles/14K NMP-ops per second on 2.66GHz core-i7 processor for RL disabled configurations, and 11K cycles/20.7K NMP-ops per second on the same machine for RL enabled configurations. Even if conservatively we consider NMP-op are similar to other X86/RISC-V instructions (which is not, as we do cycle accurate simulation for memory), gem5 approximately commits only 4K instructions per second [197].

5.7 Summary

This paper introduces MC^{2}sim, a cycle-level simulator to satisfy new demands of memory-centric computing research. MC^{2}sim supports both chip multi-processor and network of memory components. MC^{2}sim takes advantage of full-system simulators and application-level simulators, while avoiding the deficiencies of both. MC^{2}sim enables architects to perform detailed and holistic research on emerging memory-centric NMP architectures. Using MC^{2}sim, we explored different NMP techniques and compared across them. We also conduct several experiments to validate our design and ensure its correctness and desired accuracy.

6. CONCLUSIONS

The primary contribution of this dissertation is designing an ecosystem (system-on-chip) that consists of numerous independently developed chips on the same silicon board, free of communication hazards, supporting memory-centric computation for near-memory processing (NMP), integrated with a generic RL framework for minimizing the NMP operation cost. The secondary contribution is to develop a simulation tool that models the above mentioned system, capable of running the simulation at high speed, keeping the accuracy in the acceptable level for facilitating research in this area.

6.1 Dissertation Summary

Network deadlock issue has been one of the high priorities for any network designer. However, large modular SoC being in its infancy, the deadlock that occur in modular SoCs has been recently detected by Modular Turn Restriction paper and provided a solution for a particular set of routing functions adopted for each of the chiplets and SoC independently. In this study we expanded the scope to any routing algorithm and any topology offering high level of flexibility and opportunity for optimizations for the different chiplet and SoC designers. Our remote injection control technique ensures no protocol change and only change required in the network-interface of all the routers in the chiplet and only minor alteration in router stages for a class of packets. Since it can be incorporated only by changing hardware it should be easily accommodated in the design to guarantee deadlock freedom.

The deadlock avoidance not only makes the system reliable, but also improves the cost of overall operations, which otherwise be incurred by affecting the ability of the system to function without interruption or forcing the system run in a lower speed for minimizing the frequency of deadlock occurrences. Hence the deadlock avoidance allows the SoCs to exploit the network at its fullest. However, while running NMP operations in the memory network, one of the prime source of the cost of operations being the network itself, the placement of the data and computation

event is very important. In addition, the secondary impact is observed on the row-buffer hit-rate because of the scheduling computations. We utilize Reinforcement Learning techniques as an approximate solution for optimization of the data and computation mapping problem. We project our technique as a plug-and-play module to be integrated with diverse NMP systems. In this not only the placement is important, but also its timing plays huge role for achieving desired performance boost over the baseline performance.

This dissertation also introduces MC^{2}sim, a cycle-level simulator to satisfy new demands of memory-centric computing research. MC^{2}sim supports both chip multi-processor and network of memory components. MC^{2}sim takes advantage of full-system simulators and application-level simulators, while avoiding the deficiencies of both. MC^{2}sim enables architects to perform detailed and holistic research on emerging memory-centric NMP architectures. Using MC^{2}sim, we explored different NMP techniques and compared across them. We also conduct several experiments to validate our design and ensure its correctness and desired accuracy.

6.2 Future Directions

Large scale modular Systems-on-Chip (SoCs) are emerging as a promising solution for reducing data movement across different systems for big data applications, where we mostly focus on the SoC network deadlock issue and cost of the operations in this dissertation. In the due process we discover several new avenues in research that has potential to improve the large modular SoCs further.

6.2.1 SoC Interposer Network

We observe that the interposer network may face large traffic volume, as several computation chiplets are connected with memory through the active interposer. For instance, if the CPU, GPU and neural accelerators are connected with the DRAM and NVM memory through the interposer network, then all the operation offloading and data movement, along with coherence packets are going through the interposer network. With the increase in the number of chiplets connected, the pressure on the interposer network also tend to increase. High bandwidth switches like NVSwitch

may seem to be an easy fix, which also has limit on the peak bandwidth. In addition, DGX kind of switches require hierarchical network of switches in the interposer which sounds very challenging from the implementation perspective in the interposer at this point. However, new topological solution may emerge in future. In addition, the network protocol stack for the interposer network should consists of all possible protocols that chiplets attached to it may implement. Researchers may need to come up with more durable and smarter solutions, better than just staggering protocols in the network driver that has to be processed using another powerful processor.

6.2.2 Memory-centric Computing and AIMM

Scheduling NMP-ops in the finest granularity comes at the cost of offloading bandwidth bottleneck, which can be alleviated without using complex kernel level scheduling, by applying simple stride-based vectorization (variable size and not limited to cache block [21] or page-level [92] offloading) of the NMP operations, facilitated by the compiler. The stride needs to be observed among the consecutive operand addresses. We observe very low ($< 10\%$) (*BP*, *LUD*, *MAC*, *PR*, *RD*) and moderate to high (*KM*, *RBM*, *SC*, *SPMV*) scope for vectorization potentially improving up to 60% offloading bandwidth. Variable length vector NMP-ops will bring new challenges for MC NMP-op scheduling policy, which definitely needs in depth investigation. Expanding AIMM for facilitating data and command mapping for *Processing Using Memory* may alleviate the burden of handpicking different mappings for different underlying hardware, and further optimize its performance by continuously improving the mapping in orderly fashion.

Energy consumption is one of the most valuable commodity, which needs to be controlled as much as possible. The trade-off can be learnt by the RL agent and realized its actions in the underlying hardware. This process should aim to find a trade-off between the system performance and dynamic energy consumption. In the similar direction, the RL accelerator design may be simplified further to obtain cheaper inference and minimal training costs.

6.2.3 MC^{2}sim

Since MC^{2}sim is still under development, there are a lot of scope for further improvement. We just point out a broad aspect of tile based HBM-PIM network, where a detailed design of SMART-NIC and SMART-Switch have to be developed and verified for latency and bandwidth. We can think about some prefetching in the NMP system for improving the NMP performance further. An integrated RL accelerator may improve the simulation accuracy further at the cost of simulation speed. Hence, it is worth exploring the design trade-off. Energy estimation being one of the prime concerns, integrated energy estimator would facilitate the users with more certainty regarding the design viability.

REFERENCES

[1] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[2] D. Stow, Y. Xie, T. Siddiqua, and G. H. Loh, "Cost-effective design of scalable high-performance systems using active and passive interposers," in *Proceedings of the 36th International Conference on Computer-Aided Design*, pp. 728–735, IEEE Press, 2017.

[3] D. Green, "Common heterogeneous integration and ip reuse strategies (chips)," *DARPA DARPA-BAA-16-62*.

[4] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*, pp. 546–558, IEEE, 2015.

[5] S. S. Iyer, "Heterogeneous integration for performance and scaling," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 6, no. 7, pp. 973–982, 2016.

[6] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," 1988.

[7] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib, N. Jerger, and G. H. Loh, "Modular routing design for chiplet-based systems," in *International Symposium on Computer Architecture*, IEEE, 2018.

[8] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE transactions on parallel and distributed systems*, vol. 4, no. 12, pp. 1320–1331, 1993.

[9] A. Ramrakhyani, P. Gratz, and T. Krishna, "Synchronized progress in interconnection network (SPIN): A new theory for deadlock freedom," in *International Symposium on Computer Architecture*, IEEE, 2018.

[10] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, J. Labarta, and C. Minkenberg, "On-the-fly adaptive routing in high-radix hierarchical networks," in *Parallel Processing (ICPP), 2012 41st International Conference on*, pp. 279–288, IEEE, 2012.

[11] J. Flich, P. Lopez, M. P. Malumbres, J. Duato, and T. Rokicki, "Applying in-transit buffers to boost the performance of networks with source routing," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1134–1153, 2003.

[12] S. Thomas, C. Gohkale, E. Tanuwidjaja, T. Chong, D. Lau, S. Garcia, and M. B. Taylor, "Cortexsuite: A synthetic brain benchmark suite.," in *IISWC*, pp. 76–79, 2014.

[13] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "CRONO: A Benchmark Suite for Multithreaded Graph Algorithms Executing on Futuristic Multicores," in *International Symposium on Workload Characterization (IISWC)*, pp. 44–55, IEEE Computer Society, 2015.

[14] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[15] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[16] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in *Hot Chips 23 Symposium (HCS)*, pp. 1–24, IEEE, 2011.

[17] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, *et al.*, "25.2 a 1.2 v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 432–433, IEEE, 2014.

[18] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *International Symposium on Computer Architecture (ISCA)*, pp. 336–348, IEEE, 2015.

[19] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *International Symposium on Computer Architecture (ISCA)*, pp. 105–117, IEEE, 2015.

[20] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 457–468, 2017.

[21] J. Huang, R. R. Puli, P. Majumder, S. Kim, R. Boyapati, K. H. Yum, and E. J. Kim, "Active-Routing: Compute on the Way for Near-Data Processing," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 674–686, IEEE, 2019.

[22] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *International Symposium on Computer Architecture (ISCA)*, pp. 204–216, IEEE Press, 2016.

[23] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[24] G. Kim, J. Kim, J. H. Ahn, and J. Kim, "Memory-Centric System Interconnect Design with Hybrid Memory Cubes," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 145–156, IEEE Press, 2013.

[25] J. Zhan, I. Akgun, J. Zhao, A. Davis, P. Faraboschi, Y. Wang, and Y. Xie, "A Unified Memory Network Architecture for In-Memory Computing in Commodity Servers," in *International Sympoium on Microarchitecture (MICRO)*, pp. 1–14, IEEE, 2016.

[26] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pp. 32–41, 2000.

[27] B. Akin, F. Franchetti, and J. C. Hoe, "Data reorganization in memory using 3d-stacked dram," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3S, pp. 131–143, 2015.

[28] Y. Liu, X. Zhao, M. Jahre, Z. Wang, X. Wang, Y. Luo, and L. Eeckhout, "Get out of the valley: power-efficient address mapping for gpus," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 166–179, IEEE, 2018.

[29] G. Piccoli, H. N. Santos, R. E. Rodrigues, C. Pousa, E. Borin, and F. M. Quintão Pereira, "Compiler support for selective page migration in numa architectures," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, (New York, NY, USA), p. 369–380, Association for Computing Machinery, 2014.

[30] B. Goglin and N. Furmento, "Enabling high-performance memory migration for multithreaded applications on linux," in *2009 IEEE International Symposium on Parallel Distributed Processing*, pp. 1–9, 2009.

[31] M. Chiang, S. Tu, W. Su, and C. Lin, "Enhancing inter-node process migration for load balancing on linux-based numa multicore systems," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 02, pp. 394–399, 2018.

[32] Z. Duan, H. Liu, X. Liao, H. Jin, W. Jiang, and Y. Zhang, "Hinuma: Numa-aware data placement and migration in hybrid memory systems," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pp. 367–375, 2019.

[33] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 185–195, IEEE, 2013.

[34] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 58–70, IEEE, 2020.

[35] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 383–396, IEEE, 2018.

[36] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 372–385, IEEE, 2020.

[37] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 288–301, IEEE, 2017.

[38] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "Dracc: a dram based accelerator for accurate cnn inference," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[39] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, *et al.*, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 350–352, IEEE, 2021.

[40] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti, "3d-stacked memory-side acceleration: Accelerator and system design," in *Workshop on Near-Data Processing (WoNDP)(Held in conjunction with MICRO-47)*, Citeseer, 2014.

[41] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 105–117, 2015.

[42] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *International Symposium on Microarchitecture (MICRO)*, pp. 273–287, ACM, 2017.

[43] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 236–249, 2019.

[44] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[45] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE, 2014.

[46] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 279–291, IEEE, 2019.

[47] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *arXiv preprint arXiv:1611.01779*, 2016.

[48] H. Van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 272–279, IEEE, 2007.

[49] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[50] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *arXiv preprint arXiv:1803.02329*, 2018.

[51] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 39–50, 2008.

[52] K. Anjan and T. M. Pinkston, "An efficient, fully adaptive deadlock recovery scheme: Disha," in *ACM SIGARCH Computer Architecture News*, vol. 23, pp. 201–210, ACM, 1995.

[53] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, 1993.

[54] 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*, pp. 203–214, IEEE, 2008.

[55] S. Ma, N. Enright Jerger, and Z. Wang, "Dbar: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3, pp. 413–424, 2011.

[56] J. Duato and T. M. Pinkston, "A general theory for deadlock-free adaptive routing using a mixed set of resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pp. 1219–1235, 2001.

[57] M. Ebrahimi and M. Daneshtalab, "Ebda: A new theory on design and verification of deadlock-free interconnection networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 703–715, 2017.

[58] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 367–379, IEEE Press, 2016.

[59] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Transactions on parallel and distributed systems*, vol. 11, no. 7, pp. 729–738, 2000.

[60] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pp. 77–88, IEEE, 2008.

[61] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA '92, pp. 278–287, 1992.

[62] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pp. 298–309, IEEE, 2011.

[63] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, "An abacus turn model for time/space-efficient reconfigurable routing," in *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 259–270, ACM, 2011.

[64] C. Carrion, R. Beivide, J. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *High-Performance Computing, 1997. Proceedings. Fourth International Conference on*, pp. 322–329, IEEE, 1997.

[65] X. Canwen, Z. Minxuan, D. Yong, and Z. Zhitong, "Dimensional bubble flow control and fully adaptive routing in the 2-d mesh network on chip," in *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on*, vol. 1, pp. 353–358, IEEE, 2008.

[66] B. Roscoe, "Routing messages through networks: an exercise in deadlock avoidance," 1987.

[67] V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu, "Adaptive bubble router: a design to improve performance in torus networks," in *Proceedings of the 1999 International Conference on Parallel Processing*, pp. 58–67, Sep. 1999.

[68] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. Prellezo, "The adaptive bubble router," *Journal of Parallel and Distributed Computing*, vol. 61, no. 9, pp. 1180–1208, 2001.

[69] J. Flich, M. P. Malumbres, P. López, and J. Duato, "Performance evaluation of a new routing strategy for irregular networks with source routing," in *Proceedings of the 14th International Conference on Supercomputing*, ICS '00, (New York, NY, USA), pp. 34–43, ACM, 2000.

[70] L. Chen and T. M. Pinkston, "Worm-bubble flow control," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 366–377, IEEE, 2013.

[71] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger, "Leaving one slot empty: Flit bubble flow control for torus cache-coherent nocs," *IEEE Transactions on Computers*, vol. 64, pp. 763–777, March 2015.

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

[73] 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 *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 458–470, IEEE, 2014.

[74] E. Beyne and A. Manna, "High-bandwidth chip-to-chip interfaces: 3d stacking, interposers and optical i/o," in *IMEC technology forum, Taiwan*, 2013.

[75] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module gpus for continued performance scalability," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 320–332, ACM, 2017.

[76] "3DIC." `www.3dic.org/2.5D_integration`.

[77] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, pp. 481–492, 2017.

[78] Z. Sura, A. Jacob, T. Chen, B. Rosenburg, O. Sallenave, C. Bertolli, S. Antao, J. Brunheroto, Y. Park, K. O'Brien, and R. Nair, "Data Access Optimization in a Processing-in-Memory System," in *International Conference on Computing Frontiers (CF)*, p. 6, ACM, 2015.

[79] B. Hong, G. Kim, J. H. Ahn, Y. Kwon, H. Kim, and J. Kim, "Accelerating Linked-List Traversal through Near-Data Processing," in *International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pp. 113–124, IEEE, 2016.

[80] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the Impact of 3D-Stacked Memory + Logic Devices on MapReduce Workloads," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 190–200, 2014.

[81] A. F. Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," in *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 283–295, 2015.

[82] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The mondrian data engine," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pp. 639–651, 2017.

[83] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems," in *International Symposium on Microarchitecture (MICRO)*, pp. 1–13, IEEE, 2016.

[84] J. Ahn, S. Yoo, and K. Choi, "AIM: Energy-Efficient Aggregation inside the Memory Hierarchy," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 4, p. 34, 2016.

[85] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C.-Y. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H. Moreno,

J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, and Z. Sura, "Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems," *IBM Journal of Research and Development*, vol. 59, no. 2/3, p. 17, 2015.

[86] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1–14, ACM, 2018.

[87] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kusela, A. Knies, P. Ranganathan, *et al.*, "Google workloads for consumer devices: mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 316–331, ACM, 2018.

[88] V. Seshadri and O. Mutlu, "Simple operations in memory to reduce data movement," in *Advances in Computers*, vol. 106, pp. 107–166, Elsevier, 2017.

[89] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise and and or in dram," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[90] A. Yazdanbakhsh, C. Song, J. Sacks, P. Lotfi-Kamran, H. Esmaeilzadeh, and N. S. Kim, "In-dram near-data approximate acceleration for gpus," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, pp. 1–14, 2018.

[91] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "ipim: Programmable in-memory image processing accelerator using near-bank architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 804–817, IEEE, 2020.

[92] J. Huang, P. Majumder, S. Kim, T. Fulton, R. R. Puli, K. H. Yum, and E. J. Kim, "Computing en-route for near-data processing," *IEEE Transactions on Computers*, vol. 70, no. 6,

pp. 906–921, 2021.

[93] T. V. Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," in *International Symposium on Computer Architecture (ISCA)*, pp. 256–266, IEEE, 1992.

[94] G. F. Pfister and V. A. Norton, ""Hot Spot" Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. c-34, no. 10, pp. 943–948, 1985.

[95] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "IncBricks: Toward In-Network Computation with an In-Network Cache," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 795–809, ACM, 2017.

[96] S. Ma, N. E. Jerger, and Z. Wang, "Supporting Efficient Collective Communication in NoCs," in *High Performance Computer Architecture (HPCA)*, pp. 1–12, IEEE, 2012.

[97] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer-Designing an MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers*, vol. c-32, no. 2, pp. 175–189, 1983.

[98] D. K. Panda, "Global Reduction in Wormhole k-ary n-cube Networks with Multidestination Exchange Worms," in *International Parallel Processing Symposium (IPPS)*, pp. 652–659, 1995.

[99] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The IBM Blue Gene/Q Interconnection Network and Message Unit," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–10, IEEE, 2011.

[100] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proceedings of the Twenty-Third Inter-*

national Conference on Architectural Support for Programming Languages and Operating Systems, pp. 461–475, ACM, 2018.

[101] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson, "Hoard: A scalable memory allocator for multithreaded applications," *SIGPLAN Not.*, vol. 35, p. 117–128, Nov. 2000.

[102] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," in *Memory Management* (H. G. Baler, ed.), (Berlin, Heidelberg), pp. 1–116, Springer Berlin Heidelberg, 1995.

[103] K. C. Knowlton, "A fast storage allocator," *Commun. ACM*, vol. 8, p. 623–624, Oct. 1965.

[104] D. Bovet and M. Cesati, *Understanding The Linux Kernel*. Oreilly amp; Associates Inc, 2005.

[105] J. Corbet, "LKML: Transparent huge pages in 2.6.38," Jan. 2011.

[106] A. Panwar, S. Bansal, and K. Gopinath, "Hawkeye: Efficient fine-grained os support for huge pages," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, (New York, NY, USA), p. 347–360, Association for Computing Machinery, 2019.

[107] R. Courts, "Improving locality of reference in a garbage-collecting memory management system," *Commun. ACM*, vol. 31, p. 1128–1138, Sept. 1988.

[108] L. P. Deutsch and D. G. Bobrow, "An efficient, incremental, automatic garbage collector," *Commun. ACM*, vol. 19, p. 522–526, Sept. 1976.

[109] H. Lieberman and C. Hewitt, "A real-time garbage collector based on the lifetimes of objects," *Commun. ACM*, vol. 26, p. 419–429, June 1983.

[110] W. J. Bolosky, M. L. Scott, R. P. Fitzgerald, R. J. Fowler, and A. L. Cox, "Numa policies and their relation to memory architecture," *ACM SIGOPS Operating Systems Review*, vol. 25, no. Special Issue, pp. 212–221, 1991.

[111] "Numa locality,"

[112] A. C. Yao, "An analysis of a memory allocation scheme for implementing stacks," *SIAM Journal on Computing*, vol. 10, no. 2, pp. 398–403, 1981.

[113] C. P. Ribeiro, J. Mehaut, A. Carissimi, M. Castro, and L. G. Fernandes, "Memory affinity for hierarchical shared memory multiprocessors," in *2009 21st International Symposium on Computer Architecture and High Performance Computing*, pp. 59–66, 2009.

[114] C. Lameter, "Numa (non-uniform memory access): An overview," *Queue*, vol. 11, 07 2013.

[115] T. Baruah, Y. Sun, A. T. Dinçer, S. A. Mojumder, J. L. Abellán, Y. Ukidave, A. Joshi, N. Rubin, J. Kim, and D. Kaeli, "Griffin: Hardware-software support for efficient page migration in multi-gpu systems," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 596–609, IEEE, 2020.

[116] A. Awad, A. Basu, S. Blagodurov, Y. Solihin, and G. H. Loh, "Avoiding tlb shootdowns through self-invalidating tlb entries," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 273–287, IEEE, 2017.

[117] M. Oskin and G. H. Loh, "A software-managed approach to die-stacked dram," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pp. 188–200, IEEE, 2015.

[118] B. F. Romanescu, A. R. Lebeck, D. J. Sorin, and A. Bracy, "Unified instruction/translation/-data (unitd) coherence: One protocol to rule them all," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 1–12, IEEE, 2010.

[119] C. Villavieja, V. Karakostas, L. Vilanova, Y. Etsion, A. Ramirez, A. Mendelson, N. Navarro, A. Cristal, and O. S. Unsal, "Didi: Mitigating the performance impact of tlb shootdowns using a shared tlb directory," in *2011 International Conference on Parallel Architectures and Compilation Techniques*, pp. 340–349, IEEE, 2011.

[120] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[121] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[122] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 308–319, IEEE, 2014.

[123] T.-R. Lin, D. Penney, M. Pedram, and L. Chen, "A deep reinforcement learning framework for architectural exploration: A routerless noc case study," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 99–110, IEEE, 2020.

[124] J. Yin, S. Sethumurugan, Y. Eckert, C. Patel, A. Smith, E. Morton, M. Oskin, N. E. Jerger, and G. H. Loh, "Experiences with ml-driven design: A noc case study," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 637–648, IEEE, 2020.

[125] K. Wang, A. Louri, A. Karanth, and R. Bunescu, "Intellinoc: A holistic design framework for energy-efficient and reliable on-chip communication for manycores," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, IEEE, 2019.

[126] L. Peled, S. Mannor, U. Weiser, and Y. Etsion, "Semantic locality and context-based prefetching using reinforcement learning," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 285–297, IEEE, 2015.

[127] B. H. Ahn, P. Pilligundla, and H. Esmaeilzadeh, "Reinforcement learning and adaptive sampling for optimized dnn compilation," *arXiv preprint arXiv:1905.12799*, 2019.

[128] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 622–636, IEEE, 2020.

[129] N. Wu, L. Deng, G. Li, and Y. Xie, "Core placement optimization for multi-chip many-core neural network systems with reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 2, pp. 1–27, 2020.

[130] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "Booksim 2.0 user's guide," *Standford University*, p. q1, 2010.

[131] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE computer architecture letters*, vol. 10, no. 1, pp. 16–19, 2011.

[132] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 163–174, IEEE, 2009.

[133] D. I. Jeon and K. S. Chung, "CasHMC: A Cycle-Accurate Simulator for Hybrid Memory Cube," *IEEE Computer Architecture Letters*, vol. 16, pp. 10–13, Jan 2017.

[134] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[135] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "Mcsima+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 74–85, IEEE, 2013.

[136] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

[137] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li, "Pimsim: A flexible and detailed processing-in-memory simulator," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 6–9, 2018.

[138] C. Yu, S. Liu, and S. Khan, "Multipim: A detailed and configurable multi-stack processing-in-memory simulator," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 54–57, 2021.

[139] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "Napel: Near-memory computing application performance prediction via ensemble learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[140] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[141] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[142] K. Ghose *et al.*, "Marssx86: Micro architectural systems simulators," *ISCA Tutorial Session*, 2012.

[143] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pp. 33–42, IEEE, 2009.

[144] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[145] M. T. Yourst, "Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator," in *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, pp. 23–34, IEEE, 2007.

[146] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *HPCA-16*

2010 The Sixteenth International Symposium on High-Performance Computer Architecture, pp. 1–12, IEEE, 2010.

[147] J. Edler, "Dinero iv: Trace-driven uniprocessor cache simulator," *http://www. cs. wisc. edu/~ markhill/DineroIV*, 1994.

[148] P. M. Ortego and P. Sack, "Sesc: Superescalar simulator," in *17 th Euro micro conference on real time systems (ECRTS'05)*, pp. 1–4, Citeseer, 2004.

[149] G. H. Loh, S. Subramaniam, and Y. Xie, "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 53–64, IEEE, 2009.

[150] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2011.

[151] A. Jaleel, R. S. Cohn, C.-k. Luk, and B. Jacob, "Cmp$im: A binary instrumentation approach to modeling memory behavior of workloads on cmps," *University of Maryland, Tech. Rep*, 2006.

[152] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 335–344, IEEE, 2012.

[153] D.-I. Jeon and K.-S. Chung, "Cashmc: A cycle-accurate simulator for hybrid memory cube," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 10–13, 2016.

[154] D. Becker, D. William, K. Christoforos, and A. O. Oyekunle, "Efficient microarchitecture for network-on-chip routers." `http://purl.stanford.edu/wr368td5072`, 2012.

[155] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Performance Analysis*

of Systems and Software (ISPASS), 2013 IEEE International Symposium on, pp. 86–96, IEEE, 2013.

[156] AMD Research, "The AMD gem5 apu simulator: Modeling heterogeneous systems in gem5," in *In gem5 User Workshop"*, 2015.

[157] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. N. Udipi, "Simulating dram controllers for future system architecture exploration," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pp. 201–210, IEEE, 2014.

[158] AMD, "HCC sample applications, github repository, 2016." `https://github.com/ROCm-Developer-Tools/HCC-Example-Application`. Accessed: November 27, 2018.

[159] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, Ieee, 2009.

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

[161] "SPEC CPU2017." `https://www.spec.org/cpu2017`.

[162] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[163] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-aware prioritization mechanisms for on-chip networks," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 280–291, ACM, 2009.

[164] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pp. 342–353, IEEE Computer Society, 2008.

[165] 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 *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, NOCS '12, (Washington, DC, USA), pp. 201–210, IEEE Computer Society, 2012.

[166] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

[167] A. Demir, E. Çilden, and F. Polat, "Landmark based reward shaping in reinforcement learning with hidden states," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1922–1924, 2019.

[168] J. Ferret, R. Marinier, M. Geist, and O. Pietquin, "Self-attentional credit assignment for transfer in reinforcement learning," *arXiv preprint arXiv:1907.08027*, 2019.

[169] X. Guo, *Deep learning and reward design for reinforcement learning*. PhD thesis, 2017.

[170] H. Cho, P. Oh, J. Park, W. Jung, and J. Lee, "Fa3c: Fpga-accelerated deep reinforcement learning," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, (New York, NY, USA), p. 499–513, Association for Computing Machinery, 2019.

[171] J. Su, J. Liu, D. B. Thomas, and P. Y. Cheung, "Neural network based reinforcement learning acceleration on fpga platforms," *SIGARCH Comput. Archit. News*, vol. 44, p. 68–73, Jan. 2017.

[172] Z. Yan, J. Veselỳ, G. Cox, and A. Bhattacharjee, "Hardware translation coherence for virtualized systems," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 430–443, 2017.

[173] M. Plappert, "keras-rl." `https://github.com/keras-rl/keras-rl`, 2016.

[174] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[175] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[176] A. Singh, D. Carnelli, A. Falay, K. Hofstra, F. Licciardello, K. Salimi, H. Santos, A. Shokrollahi, R. Ulrich, C. Walter, *et al.*, "26.3 a pin-and power-efficient low-latency 8-to-12gb/s/wire 8b8w-coded serdes link for high-loss channels in 40nm technology," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 442–443, IEEE, 2014.

[177] D. Delorie, "Glibc wiki - overview of malloc." `https://sourceware.org/glibc/wiki/MallocInternals`.

[178] "TCMalloc : Thread-Caching Malloc." `https://google.github.io/tcmalloc/design.html`.

[179] "jemalloc." `http://jemalloc.net/`.

[180] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads," in *IEEE International Symposium on Workload Characterization (IISWC'10)*, pp. 1–11, IEEE, 2010.

[181] "Multiple features data set," *`https://archive.ics.uci.edu/ml/datasets/Multiple+Features`*.

[182] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[183] G. E. Hinton, "Boltzmann machine," *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007.

[184] "Netflix public data sets," *https://github.com/Netflix/vmaf/blob/master/resource/doc/datasets.md*.

[185] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, p. 14, 2017.

[186] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, 2009.

[187] M. Poremba, I. Akgun, J. Yin, O. Kayiran, Y. Xie, and G. H. Loh, "There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes," in *International Symposium on Computer Architecture (ISCA)*, pp. 678–690, ACM, 2017.

[188] S. Kaxiras and M. Martonosi, "Computer architecture techniques for power-efficiency," *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[189] Y. Lee, S. H. Seo, H. Choi, H. U. Sul, S. Kim, J. W. Lee, and T. J. Ham, "Merci: efficient embedding reduction on commodity hardware via sub-query memoization," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 302–313, 2021.

[190] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[191] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (IEEE Cat. No. 03TH8721)*, pp. 19–24, IEEE, 2003.

[192] "AXI reference guide," *https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf*, 2011.

[193] T. Bandyopadhyay, R. Chatterjee, D. Chung, M. Swaminathan, and R. Tummala, "Electrical modeling of through silicon and package vias," in *2009 IEEE International Conference on 3D System Integration*, pp. 1–8, IEEE, 2009.

[194] E. Naviasky, "Defining a new high-speed, multi-protocol serdes architecture for advanced nodes," *Cadence Design Systems White Paper*.

[195] A. Athavale, "High-speed serial i/o made simple a designers' guide, with fpga applications," 2021.

[196] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, *et al.*, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 43–56, IEEE, 2021.

[197] T. Ta, L. Cheng, and C. Batten, "Simulating multi-core risc-v systems in gem5," in *Workshop on Computer Architecture Research with RISC-V*, 2018.