AN ABSTRACT OF THE DISSERTATION OF

Yunfan Li for the degree of Doctor of Philosophy in Computer Science presented on April 29, 2021.

Title: Predicting and Improving Throughput, Responsiveness and Battery Life of Computer Systems by Machine Learning

Abstract approved: ____

Lizhong Chen

The Machine Learning (ML) algorithms are increasingly explored in varies of fields including designing and optimizing computer systems. Recent research, such as optimizing memory/cache prefetching by ML training or predicting traffic pattern in throughput processors, also exhibits a promising future of introducing ML into computer system design and optimization. Throughput optimization in throughput-oriented processors is imperative as the computing workload of parallel and cloud computing have been growing rapidly in recent years. At the same time, throughput optimization can be time consuming when applying conventional design and optimizing process as the design space is prohibitively huge. In the first part of this dissertation, we firstly define a huge and complicated design space in silicon interposer-based throughput processors, then utilize Monte Carlo Tree Search model (MCTS) to exploit and explore the design space. The evaluation results show that the system performance is improved by over 20% with only 0.05% design space is assessed. Performance and power (PnP) are also the most important metrics that are utilized by original equipment manufacturers (OEMs) to conduct the design of a device. Current PnP measurements rely on manual hardware swapping and testing for systems which is time consuming and not financially-efficient. A fast and accurate PnP value prediction solution can guide OEMs to understanding the basic behaviour of different hardware components, and, more importantly, shortens the time to market of a device. In the second work, we explore the common ground between natural language processing (NLP) problems and system PnP prediction problems, and develop an NLP-like solution to resolve the problem. The solution is available to extract the inter- and intra-relationship among the existing system components and to predict the behavior of system components that have not appeared before. The results of our evaluation demonstrate that the solution achieves as high as 94% labeling accuracy in a real-measured dataset.

Predicting and Improving Throughput, Responsiveness and Battery Life of Computer Systems by Machine Learning

by

Yunfan Li

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented April 29, 2021
Commencement June 2021

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

Machine Learning (ML) algorithms have been increasingly exploring in various fields, such as image classification, speech recognition, AI for video games *etc*. ML solutions have also been drawing more attention in computer system design and optimization communities to optimize memory/cache systems [58, 10, 51, 53], improve branch prediction accuracy [24, 23, 50] or reduce system-level energy consumption [55, 42, 5]. There are some other works that focus on improving the throughput of networks-on-chip (NoCs) [45, 9] by applying ML-based traffic allocation strategy or routing policy. In general, machine learning is promising and capable of resolving different types of design and optimization issues in the field of computer systems.

Throughput-oriented processors (e.g., GPUs) have been increasingly used to speed up a wide range of conventional and emerging applications. Due to the large number of cores in the processors, NoCs have been gaining significant research interest [7, 6, 61, 19, 27, 44, 60, 59] to provide low-latency and high-throughput on-chip communication. Meanwhile, with the recent advent of silicon interposer and 2.5D integration technology, memory chips can be integrated with the processor chip in one package to provide dramatically increased memory throughput [52, 37, 12, 20]. However, this places a huge pressure on the NoC component. Unfortunately, existing schemes have turned out to be ineffective when applied to these interposer-based systems. With more advanced interposer technologies on the horizon [17, 21, 8, 32], the performance gap between the

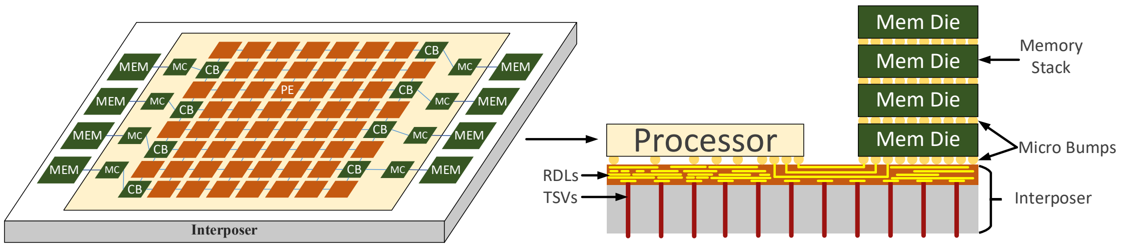


Figure 1.1: Overview and cross-section view of an interposer-based throughput-oriented processor.

memory and NoC will likely get even widened. Thus, it is imperative to redesign the NoC accordingly to meet the requirements of interposer-based throughput processors.

Silicon interposers bring both major problems and opportunities to throughput-oriented processors. On the one hand, the many-to-few-to-many traffic pattern in those processors may cause a bottleneck in the reply network, where data reply packets that are destined to the many cores are injected through only a few injection nodes. With the boosted memory bandwidth in stacked memory in interposer systems, this injection bottleneck is greatly intensified and is only getting worse with future memory technologies. On the other hand, interposer contains multiple dedicated Redistribution Layers (RDLs), which provides abundant wiring resources that are currently underutilized [21, 35]. These wiring resources have electrical characteristics that are similar to that of on-chip links [46]. This provides a basis for exploiting a hybrid use of on-chip and interposer components.

In the first work, we explore the new wiring opportunities brought by the interposer to address the intensified injection bottleneck that is also caused by the interposer. Given the root cause of the injection bottleneck is the few-to-many traffic pattern, we propose *Equivalent Injection Routers (EIRs)* that transform the traffic to many-to-many pattern, thus fundamentally solving the bottleneck problem. This is achieved by providing each injection point with a group of equivalent injection routers, all of which have "equivalent" capability in terms of accepting and distributing the injecting traffic. The required additional interconnects are provided by the redistribution layers in the interposer. While the concept of EIRs is straightforward, selecting the set of equivalent routers requires comprehensive consideration from topological, architectural and physical aspects. This leads to a large design space as explained later in Section 3.

To demonstrate the feasibility of the proposed EIR approach, we have developed *EquiNox* as a design example. The scheme employs a N-Queen based cache bank placement as the basis for selecting equivalent routers. As the solutions of N-Queen are not unique, a scoring policy is developed to select the placement that minimizes network congestion and maximizes EIR potential. The groups of EIRs are then selected by a carefully designed Monte Carlo Tree Search (MCTS) method to search through the design space, while balancing the number of EIRs, their impact on the network, the needed interposer links, the length of the links, and the number of cross-points in the RDLs. The network interface architecture is also enhanced to support the increased injection flexibility from EIRs. The proposed EquiNox is able to meet the requirements of topologically equivalent, architecturally efficient and physically viable EIR designs. Evaluation on a wide range of benchmarks shows that EquiNox achieves 47.7% reduction in execution time and 55.0% reduction in energy-delay product (EDP) compared with a single network scheme, and 23.5% reduction in execution time and 32.8% reduction in EDP compared with a separate network scheme.

Performance and power (PnP) are the most important design metrics for OEMs to develop a device. In recent years, with the proliferation of mobile device, the battery life becomes a major consideration for customers. However, the techniques that prolong the battery life usually worsen the performance of a device. As the performance is also an important feature in designing process, the trade-off between power and performance becomes rather difficult to be balanced. Therefore, the OEMs usually continuously test and tune a device until the pre-determined PnP target is satisfied. To measure the PnP values for various systems, human experts have to manually swap hardware components every time, and then test PnP values for the new system. Such process requires massive number of human experts and consumes significant amount of time. This is not financially-efficient and time-efficient for platform design companies like Intel or OEMs like Dell. Therefore, a PnP prediction solution is necessary to accelerate the process and to reduce the time to market of a device.

In this work, we propose a natural language processing (NLP)-like solution which includes a word vector interpreter model, a hardware representing model, and a PnP prediction model. The hardware components of a system are firstly transformed from string data to floating point vectors by the word vector interpreter model. This is done by an algorithm called Continuous Bag Of Words (CBOW). Ordinary a CBOW model is designed for only capturing the context relationship of a sentence. To better fit CBOW to our work, we enhance it by integrating the measured PnP value of a system as an additional output target. For the unseen components, we predict their word vectors by utilizing the the hardware representing model with the component specs. Eventually, the word vectors of a system are fed into the PnP prediction model to obtain the PnP values. Our evaluation demonstrates that the PnP prediction model accuracy is around 91.7%, and the labeling accuracy is as high as 94.5%.

The rest of the thesis is organized as follows. Chapter 2 summarizes the related work. Chapter 3 introduces the MCTS-based throughput processor optimization in detail. Chapter 4 proposes our NLP-like solution to predict PnP values for the systems with unseen hardware components. Finally, chapter 5 concludes the thesis.

Chapter 2: Related Work

There is a substantial amount of research that focuses on applying machine learning to computer system design and optimization. One such topic concerning system optimization would be branch prediction, which is naturally a problem that can be resolved by supervised learning. Jimenez et al. [24] propose a perceptron-based branch predictor to achieve higher prediction accuracy than a conventional two-level predictor, and later Jimenez [23] optimizes the proposed predictor by integrating a set of novel techniques such as a dynamic learning threshold. Alternatively, a CNN based approach is applied towards branch prediction by Tarsa et al. [50], and they propose a "CNN-Helper" predictor to further improve the prediction accuracy and show the efficiency on hard-to-predict branches. Prefetching is also a widely explored field of applying machine learning. Zeng and Guo [58] propose a LSTM-based memory prefetcher that achieves better prefetching efficiency than traditional approaches. Braun and Litz [10] train a LSTM on several microbenchmarks which characterized the memory access patterns and provided a better understanding of the relationship between model parameters and the memory access patterns. The cacheline reusing policy can also be predicted by perceptrons [51] or a decision tree [53]. Networks-on-Chip (NoCs) is another major field where researchers have explored machine learning applications. A neural network predictor is proposed by Reze et al. [45] to manage global resource allocation and per-node DVFS. Boyan and Littman [9] design a Q-learning based routing policy to improve the packet routing efficiency.

Their evaluation results suggest that the proposed Q-learning policy distributes the on-chip traffic more evenly. There are many other works [55, 42, 5, 36, 38] employ different machine learning models such as artificial neural network or reinforcement learning models on optimizing system-level energy consumption to achieve energy-efficiency system design.

Bakhoda *et al.* observe the M2F2M traffic pattern and propose a scheme specifically for this pattern [6]. Some works propose to split a reply network into several reply subnetworks physically [27, 60] or even in time division fashion [44] to gain a higher injection rate for the reply network. Jang *et al.* propose VC-Monopolization design for single network NoC system in GPGPUs, which improves VC utilization and network throughput. These works have been compared in Section 6. The design space in interposer-based 2.5D system is exploited for NoCs to build energy-efficent NoCs for many-core CPUs [21, 25]. In [21], some general cases of designing energy-efficient NoC are explored. In [25], an efficient design of NoCs in chiplet interposer-based CMP systems is proposed. However, those designs have limited effectiveness when adopted directly to interposer-based throughput processors due to significantly different traffic patterns.

Express links have been proposed for both off-chip (e.g., Express Cube [14]) and on-chip networks (e.g., Flattened Butterfly [28] and MECS [16]). Those topologies typically use an extensive number of express links, which are redundant in throughput processors due to minimal inter-PE traffic.

Another approach to mitigate the impact of the injection bottleneck is to compress reply packets to smaller packets, and unzip after after they are injected [29]. Also, it is possible to alleviate injection bottleneck by employing near-data processing in interposer-based systems [18]. With near memory computing nodes in memory stacks, the reply traffic that needs to travel through the reply network is reduced. These two schemes are largely orthogonal and complementary to the our proposed EquiNox. Additionally, Ziabari *et al.* propose an asymmetric network design in the request and reply network where unnecessary links are removed to exploit memory traffic characteristics in GPUs [61]. Zhao *et al.* design a Heterogeneous Ring-Chain network (HRCnet) for the reply network which provides lower area and power consumption and reduces packet conflicts with a ring-based topology [59]. While these works are effective in their targeted contexts, they do not consider interpose-related characteristics.

Chapter 3: Throughput Optimization by Reinforcement Learning

3.1 Background and Motivation

3.1.1 Interposer-based Throughput Processors

To meet the growing demand of high-bandwidth and low-latency memory in many-core processors, 2.5D integration systems have been proposed and commercially yielded to achieve wafer-level integration of processor dies and memory dies. Figure 1.1 depicts the architecture of a typical silicon interposer-based throughput processor. The interposer is a silicon substrate that provides mechanical and electrical characteristics to integrate multiple dies [21]. On the left side of Figure 1.1, the processor die is represented by the large light yellow square. Eight memory dies are located outside the processor die, denoted as the green squares labeled with “MEM". In the processor die, processing elements (PEs) and last level cache banks (LLCBs or simply CBs) are placed in a tile-based fashion. Each cache bank is connected with a dedicated memory controller (MC) that interfaces with a memory die. Each memory die is actually a die stack, which is composed of several individual dies that are vertically placed on top of each other to form a 3D die stack. This technology is known as the High Bandwidth Memory (HBM) [1, 39]. HBMs can greatly benefit from integration in 2.5D systems, since the in-package interconnects offered by the interposer have superior physical properties than off-chip interconnects.

In interposer-based processors, the MCs are usually located near the edge of the processor die to ensure short and fast interposer connections with the memory stacks. As each MC is connected with a CB, only a few CBs exist that are shared by all the PEs. While the locations of the MCs are less flexible, the placement of CBs can be adjusted to achieve more efficient on-chip communication. Different from the conventional many-core CPUs, the PEs in throughput processors (e.g., Stream Multiprocessors (SMs) in GPUs) have little inter-PE communication, but instead communicate with CBs (and then memory stacks) directly. PE-generated request packets are sent to CBs through a *request network*. Reply packets can be generated directly by the CBs in case of cache hits. Otherwise, the CBs will first fetch data from the memory and then generate reply packets containing the data. The reply packets are sent back to the PEs through a *reply network*. This traffic flow from the many PEs to a few CBs and then back to the many PEs is commonly referred to as the Many-to-Few-to-Many (M2F2M) traffic pattern in throughput-oriented processors [6, 3].

Figure 1.1 also shows the cross-section view of the silicon interposer structure. The processor die and memory stacks are integrated with the interposer via micro-bumps (μbumps) in a face-down fashion (flip-chip packaging technology)[37, 15, 17]. For example, when integrating the processor die with the interposer layer, the die is first flipped, so the surface of the chip is facing down and attached to the interposer. As a result, μbumps consume chip surface area. To connect the processor die with the memory dies, wires are routed through the interposer layer with support of the μbumps. Each wire must have a corresponding μbump to provide the electrical connectivity [37, 17, 21]. Wires in the interposer layer are implemented by fine-pitched metal layers called Redistribution

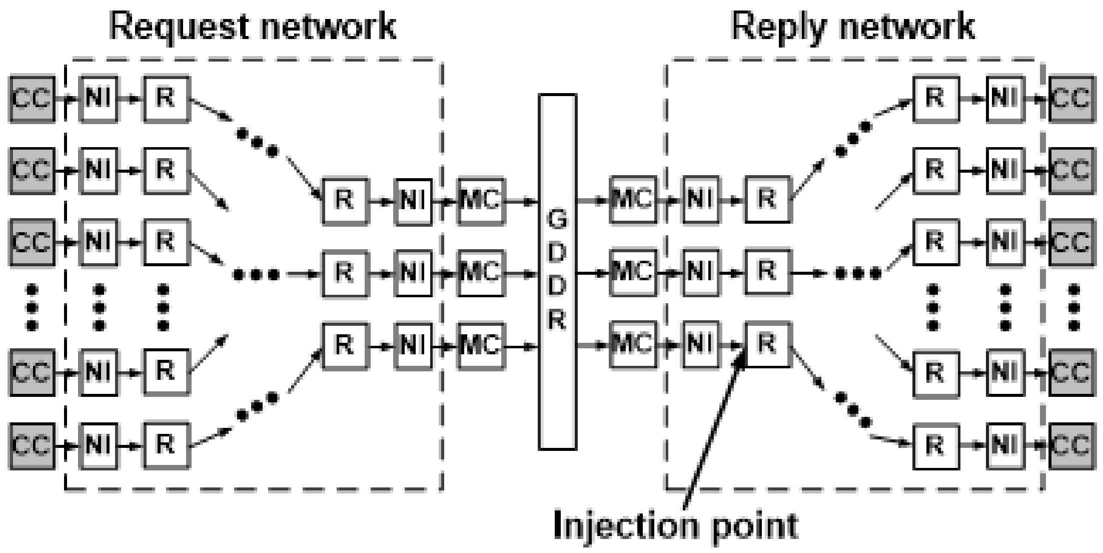


Figure 3.1: End-to-end traffic flow (CCs on both sides are the same).

Layers (RDLs) that provide high-bandwidth and low-latency interconnections [12]. To achieve that, the material of the RDLs is usually copper instead of aluminum for lower resistivity and better scalability (higher wire density). However, the Damascene process [54] needs to be used to deal with the poor oxidation and corrosion resistance of copper. In practice, to yield sub-micro pitch metal layers, interposer RDLs may employ a more complex dual-damascene process [34]. Below RDLs, Through-Silicon Vias (TSVs) are used to transfer electrical signals from RDLs to the outside of the package[1].

3.1.2 Understanding GPGPU NoC Bottleneck

In this section, we present a set of experiments which, collectively, study the interactions among NoC components in GPGPUs to increase the understanding of potential bottlenecks

[1]The C4 bumps at the bottom of the interposer layer are not shown in the figure for better clarity.

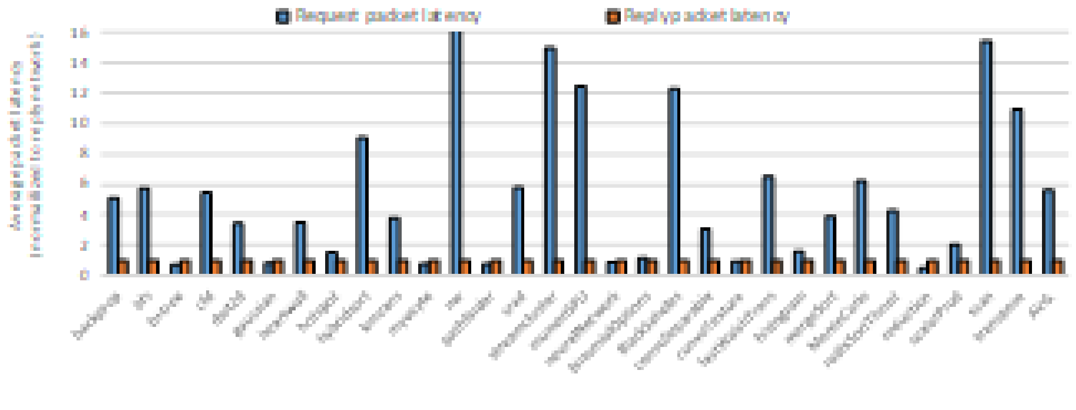


Figure 3.2: Request vs. reply packet latency.

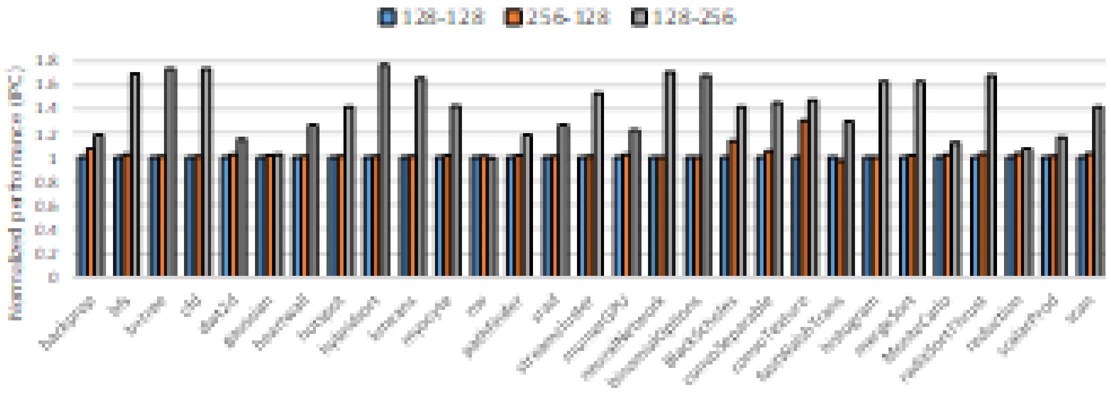


Figure 3.3: Impact of changing request-request link widths.

in the network. We use GPGPU-Sim[7] and BookSim 2.0[22] with a wide range of representative workloads from the Rodinia[13] benchmark suite and CUDA SDK[40].

First, on the high level, the NoCs in the end-to-end flow in Figure 3.1 can be divided into a request network and a reply network (denoted by the dotted squares in the figure). We use results in Figures 3, 4 and 5 to show that, due to the backpressure from the reply network to the request network, the bottleneck of GPGPU NoC is on the reply network side.

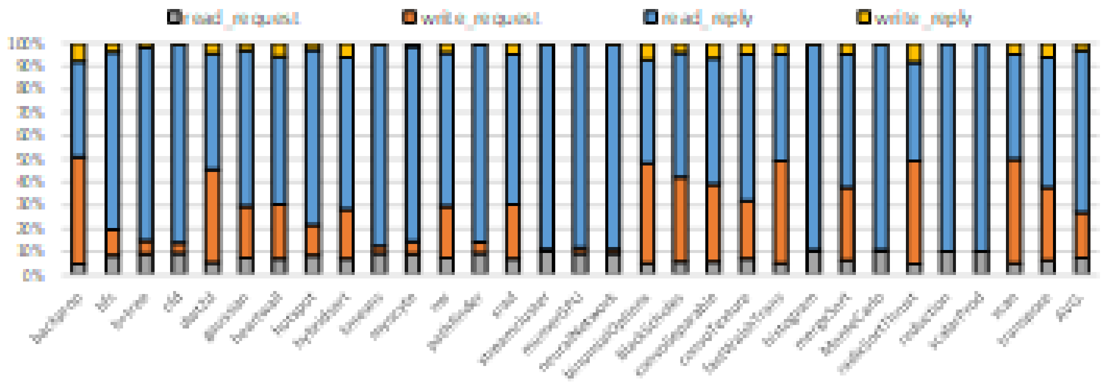


Figure 3.4: Relative percentage of 4 packet types.

Figure 3.2 compares the average packet latency of the re-quest and reply network for a typical GPGPU configured with 28 compute nodes, 8 MC nodes, and 128-bit NoC link width. At the first glance, with the request packet latency being 5.6X of the reply packet latency, on average, it seems that the bottleneck is on the request network side. However, when we double the link width of the request network from 128-bit to 256-bit, the average IPC only increases by 0.8%, as shown in Figure 3.3. In contrast, if the reply network link width is doubled, a 25.6% increase in the average IPC is observed. This indicates that the reply network is the actual limiting factor.

To explain why the reply network is more congested, Figure 3.4 examines the relative percentage of four packet types that coexist in GPGPU NoCs, taking into account the number of flits each packet type has to accurately reflect the network traffic load. Among the four packet types, read-request and write-reply are typically short packets; whereas read-reply and write-request are long packets with multiple flits, as they contain large chunk of data. Although each read (write) request packet in the request network

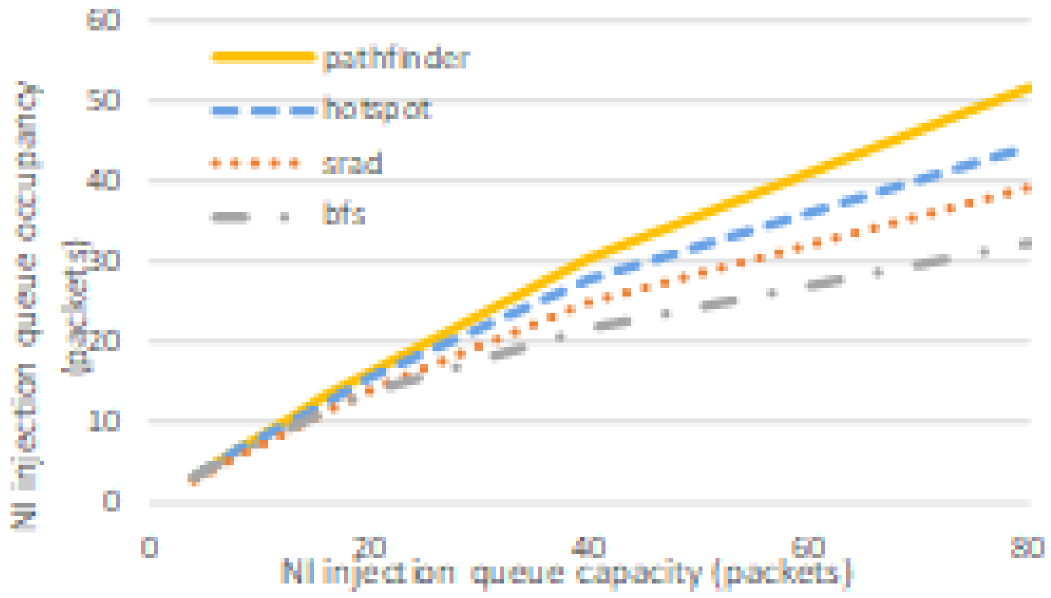


Figure 3.5: NI injection queue occupancy.

corresponds to exactly one read (write) reply packet in the re-ply network, there are considerably more read transactions than write transactions in most of the benchmarks as shown in the figure. As a result, the reply network, which predominantly carries long read-reply packets, has much more traffic load than the request network (e.g., 72.7% vs. 27.3% of the total NoC traffic in Figure 3.4), thus being more prone to congestion. When the reply network is congested, its injection buffer queues in NIs would be gradually filled up (the NIs in the right half of Figure 3.1). This slows down the processing and forwarding of data at the MC nodes to the NIs which, in turn, slows down the processing of requests at the MC nodes on the request network side. Consequently, request packets start to be queued up backward across the routers in the request network, until the backpressure eventually propagates all the way back to the source CC nodes. Analogous to the parking lot problem with a congested exit point, the cars that are the farthest from the exit experience the longest waiting time. Similarly, packets in the request network experience longer NoC latency even though the congestion happens in the reply network.

It is important to note that our simulation does not artificially create the congestion in Figure 3.2 by using 128-bit NoC links. The 128-bit link width is sufficiently wide to match the memory traffic. We show this at two levels. First, at the per-MC level, each connected GDDR5 is modeled after GTX980. According to the specification [40], it operates at 1.75GHz with 32-pin and quadruple data rate. This offers up to 28GB/s of incoming data to an MC (1.75GHz × 32b × 4 = 28GB/s). Meanwhile, each NoC link can transfer up to 128b × 1GHz = 16GB/s of data from an MC. This leads to a total outgoing data from an MC to be 48GB/s (i.e., 3 links from an MC that is located on the edge) or 64GB/s (4 links for a non-edge MC), which is more than the incoming data to the MC. Second, at the aggregate level, the total incoming data from all the 8 MCs is 28GB/s × 8 = 224GB/s (again, matched with the product specification [36]). Prior work has shown that the bisection bandwidth of the NoC need to be around 80% of the total MC bandwidth [6], which is 224GB/s × 0.8 = 179.2GB/s in this case. With 128-bit links, the bisection bandwidth of the simulated NoC is 128b × 1GHz × 12 (12 uni-directional links in the bisection of a 6×6 mesh) = 192GB/s, which is larger than the needed NoC bi-section bandwidth. The above calculations indicate that there must be other factors that cause the congestion in the reply network.

Next, we examine the reply network more closely to identify which part of the network is the bottleneck that limits the traffic flow. Simulation results of running a mix of 30 benchmarks show that, the average link utilization of the reply network is actually very low, with only 0.084 flit/cycle. However, the average link utilization of the injection links (i.e., the links from NIs to the connected routers in Figure 3.1) is 0.39 flits/cycle which is more than 4.5X the utilization of the links within the reply network. This indicates that the injection points can be the potential bottleneck. To verify this is indeed the case, we intentionally increase the capacity of the injection buffer queues in the NIs and record the queue occupancy. Essentially, if the injection point is the bottleneck and limits the traffic flow, as the queue capacity increases, more reply packets would be buffered in the injection queues, waiting to be injected. Figure 3.5 plots the results and confirms that the queue occupancy closely tracks the queue capacity as it increases from the size of 4 long read-reply packets to 80 long packets. Other benchmarks exhibit similar characteristic and are omitted in the figure for clarify.

The main reason why the injection point is the bottle-neck is that the reply network has a few-to-many traffic pattern caused by the high CC-to-MC ratio (e.g., 28 vs. 8). All the reply data that needs to be distributed back to the many CC nodes is forwarded through only a few injection points. This is further worsened by the large number of long read-reply packets in the reply network, thus concentrating the already heavy traffic to only a few paths from the NIs to the connected routers. The reply injection bottleneck of GPGPU NoCs may seriously limit the maximum achievable throughput of the traffic flow. It also increases the packet latency by blocking critical reply data at the injection points that is needed by programs to make forward progress. Nevertheless, simply increasing the width of injection links is not enough to remove this bottleneck, as the injection links are not isolated components but are closely interacting with NIs on the one end and routers on the other end. Both ends are not fully capable of handling such an increased traffic from wide injection links, if conventional CPU NoC designs are used. What is needed is a matching NI architecture that can supply a fast rate of traffic to the injection points, and a matching router architecture that can consume the injected packets by quickly transferring packets out of the injection points, as proposed in this work.

3.1.3 Intensified NoC Bottleneck

While the use of silicon interposer brings many benefits, it also worsens the injection bottleneck in the on-chip network. Specifically, each PE node or CB node is associated with an on-chip router, and the routers are connected to form the NoC. A node sends/receives packets to the NoC through a network interface (NI). As mentioned, there are two networks in the NoC: a request network and a reply network. The reply network carries much heavier traffic than the request network. This is because typical workloads for throughput processors have a lot more reads than writes. Read requests are short packets, but read replies are long packets containing cache line data. Our simulation results also confirm this, showing that reply traffic (including read reply and write reply) accounts for 72.7% of the total NoC traffic in terms of bits, and only 27.3% is request traffic (read request and write request). Therefore, the reply network is more susceptible to congestion than the request network [61, 27]. As all these heavy reply traffic is eventually injected into the reply network through a few CB nodes, these injection points (CB nodes) become the performance bottleneck of the entire NoC [6].

In interposer-based systems, the injection bottleneck is intensified as the throughput of HBM is significantly higher than that of conventional DRAMs. The second generation of HBM achieves up to 256GB/s [2] which is about 10X higher than GDDR5. This dramatic

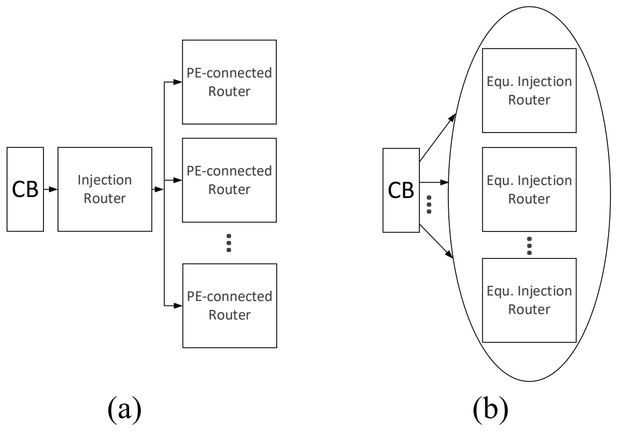


Figure 3.6: (a) Existing injection for a CB node; (b) Each CB has a group of equivalent injection routers (EIRs).

improvement in memory bandwidth has put a huge traffic pressure on the injection points in the reply network. To-date, only a few of works [6, 19, 27, 44, 60] have targeted the NoC injection bottleneck, but none of them have examined the issue in interposer-based systems where the throughput demand are drastically different. Alternatively, some works exist (e.g., [21, 25]) that utilize interposer resources to improve on-chip networks in many-core CPUs. However, as the M2F2M communication is very different from the all-to-all traffic pattern in CPUs, it is ineffective to adopt these designs for throughput processors. More discussions and quantitative comparisons are provided in later sections, but essentially, without a more specific and effective solution, the gap between the demanded data transfer rate in new memory technologies and the supported rate in current injection points will continue to be widened in the near future.

3.1.4 Interconnects Opportunity in Interposer

Although the injection bottleneck is worsened by the stacked memory and 2.5D integration, the features of interposer also open up new opportunities for interconnection. First, there is much vacant space under the processor die to route wires in the RDLs. This is because die-to-die interconnects and the associated μbumps are placed near the boundaries of the processor die and memory stack, as shown in Figure 1.1. This leaves the majority of the area under the processor die unused. Second, RDLs are composed of multiple metal layers. Although the total number of RDLs is limited due to yielding cost, substantial wiring resources are available even with the layers in the current RDLs. Third, the wire latency in the interposer is comparable to that of the die [46]. This allows a hybrid use of interposer links and on-chip links without causing concerns on unmatched signal transfer latency. Moreover, the floor planning of wires in RDLs is independent from that of the processor or memory dies (except for the interfacing μbumps), so the cross-layer wiring complexity within RDLs is not exposed to other system components.

With the above advantages, the next question is how the abundant wiring resources in the interposer can be utilized more efficiently to help with system designs, such as addressing the intensified injection bottleneck.

3.2 Equivalent Injection Routers

3.2.1 Eliminating Few-to-Many Bottleneck

The root cause for the reply injection bottleneck is a mismatch in quantity, where a *few* injection routers (i.e., CB-connected routers) need to handle all the injection traffic that is destined to the *many* PE-connected routers. Therefore, a fundamental solution to this problem is to somehow transform the “few-to-many" traffic pattern to a “many-to-many" traffic pattern. To this end, we propose the approach of *Equivalent Injection Routers (EIRs)* to eliminate the injection bottleneck. In the existing architecture as shown in Figure 3.6(a), the injection traffic from a CB is bottlenecked at the CB-connected injection router. In contrast, the proposed approach in Figure 3.6(b) provides a group of injection routers that have equivalent capability in terms of accepting and distributing the injection traffic from a given CB. Each CB has its own group of EIRs that are located strategically, so injected packets can be quickly distributed. Collectively, all the EIRs create many injection points to realize the many-to-many traffic pattern. However, implementing the idea of EIRs in conventional processors is very difficult, as it requires additional interconnects between a CB and all the EIRs in the group of that CB. This is where the RDLs in the interposer become useful. Since RDLs are underutilized and largely independent from the main NoC, RDLs can be a great resource to route the needed interconnects. In a sense, EIRs provide a nice solution that utilizes the wiring opportunities brought by interposer to address the intensified injection bottleneck that is also caused by interposer (and stacked memory) in the first place.

While the concept of EIRs looks straightforward, selecting the set of equivalent routers requires comprehensive consideration from topological, architectural and physical aspects, as explained below.

3.3 Considerations of Selecting EIRs

3.3.1 Topologically Equivalent

Every EIR in a CB's group is topologically equivalent in the sense that the CB directly connects with every EIR and can use any EIR for packet injection. Therefore, the first consideration is to determine the optimal number of EIRs in a group. At one end of the spectrum, if there is only one EIR per group, the design regresses to the existing architecture in Figure 3.6(a) where injection is congested. At the other end of the spectrum, if a group includes all the PE-connected routers as the EIRs, the CB can basically send packets to any PE in just one hop through the EIRs. In that case, however, the capacity in transferring traffic out of the CB is way higher than what the CB can possibly inject, thus leaving most of the added links (in the interposer) between the CB and EIRs idle. Therefore, the optimal number of EIRs should be determined carefully.

3.3.2 Architecturally Efficient

Another consideration is whether the selection of EIRs is architecturally efficient, even with the same number of EIRs per group. For example, with 4 EIRs in a group, there are numerous combinations that may have dramatically different architectural efficiency. Figure 3.7 presents an example where the blue group and the gray group both have 4 EIRs.

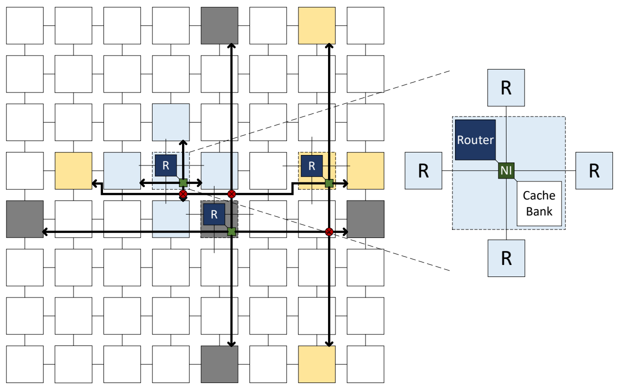


Figure 3.7: Different examples of a 4-EIR group.

As the injection routers in the blue group are located closely, the region would easily become a hot zone that leads to high queuing latency for injected packets. In contrast, the gray group distributes the EIRs further into the network, thus lowering the contention among injection traffic and being more architecturally efficient than the blue group. Due to the large number of combinations under a given number of EIRs in a group, it can be difficult to identify the optimal EIR selection in the design space. Note that, while it is beneficial to distribute EIRs across the network, the design in the gray group requires long wires and increases the probability of having wire intersection, both of which place additional constraints on physical viability, as discussed next.

3.3.3 Physically Viable

The positions of EIRs should also be selected in a way that is friendly for physical implementation. There are three main constraints due to the unique characteristics of interposer-based systems: 1) length of interposer links, 2) number of intersection points in redistribution layers (RDLs), and 3) area overhead of μbumps.

First, shorter interposer links are preferred, as long links need repeaters which would require active silicon interposers. Active interposers face greater thermal challenges and complexity than passive interposers. Consequently, EIRs cannot be positioned too far away from the CB.

Second, wire intersection in the interposer needs to be minimized, as intersection points require separate metal layers in the RDLs. For example, in Figure 3.7, at least two layers are needed to handle the three points of intersection (the three red dots). Due to the dual-damascene process, yielding complexity increases exponentially as more metal layers are included [54, 41]. The process is very costly because of the operations in cleaning residual photoresist and protecting hydrophilic low-κ dielectric films [26].

Third, because of the face-down integration in 2.5D integrated chips, μbumps consume on-chip area of the top dies, e.g., the processor die and memory dies. Every interposer wire needs a dedicated μbump to ensure electrical connectivity with other dies. Taking $40\mu m$ pitch μbumps [15] as an example, each 128-bit bi-directional link consumes around $0.34mm^2$ μbump area. This overhead can be quite substantial when the number of interposer links is large (e.g., prior works on CPU NoCs need hundreds of interposer links). Thus, it is challenging to place EIRs strategically that would require much fewer interposer links, while still being able to avoid the reply injection bottleneck.

3.3.4 Other Complications

In addition to the above three aspects, there are other factors that may potentially affect the design of EIRs. One major issue is the placement of CBs that has considerable impact

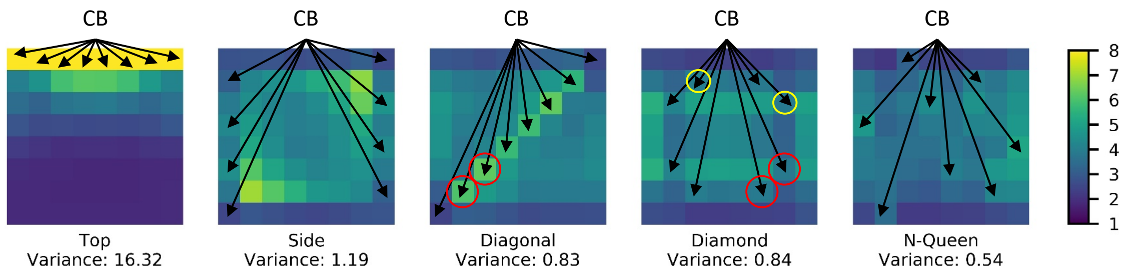


Figure 3.8: Heat map of average router travel through cycles in different CB placement.

on system performance. For instance, if multiple CBs are placed closely at the same row, the contention among injection traffic would be very high even with the help of EIRs. Also, the eight nodes surrounding a CB node have more injection traffic, so it is better not to include these surrounding nodes as EIRs which would otherwise draw even more injection traffic. All these factors, compounded by the choice of the number of EIRs in a group, the different combinations of EIRs, and the resulting length/number/intersection of interposer links, make the approach of EIRs challenging (but also interesting). In the next section, we present a design example that integrates *N*-Queen and Monte Carlo Tree Search methods to explore this large design space and materialize the benefits of EIRs.

3.4 Design Example: EquiNox

3.4.1 Overview

In this work, we have developed *EquiNox* as a case in point to demonstrate the feasibility and effectiveness of the proposed approach on using equivalent injection routers (EIRs). EquiNox contains the right combination of several design elements to meet the requirements of topologically equivalent, architecturally efficient and physically viable

EIR designs. The scheme employs a N-Queen based cache bank placement as the basis for selecting equivalent routers. Since the solutions of N-Queen are not unique, a scoring policy is developed to select the placement that minimizes network congestion and maximizes EIR potential. The actual group of EIRs is then selected by a carefully designed Monte Carlo Tree Search (MCTS) method. The proposed MCTS balances the number of EIRs and their capability in distributing the injection traffic. It also helps to determine the locations of EIRs by simultaneously reducing the number of needed interposer links, the length of the links, and the number of cross-points. Finally, a few low-cost but critical changes are applied to the NI architecture to support the increased injection flexibility that is offered by multiple equivalent routers. The following subsections describe these design elements in more detail.

3.4.2 Contention-aware CB Placement

As the basis of the EIR approach, we first present a last-level cache-bank placement that is benign to equivalent injection routers.

Hints from Existing Placements: Several popular CB placements exist, such as Top, Side, Diagonal and Diamond [3] that are originally proposed for the all-to-all traffic pattern in many-core CPUs. We analyze these placement schemes to obtain some hints for finding placements that are good for the traffic patterns in throughput processors. To get a visual intuition of the traffic situation under different placements, Figure 3.8 draws the heat map of the average number of cycles that a flit experiences when traveling through a router in the reply network (where the injection traffic forms the few-to-many traffic pattern). Each colored block denotes a router. A brighter block means that flits spend more cycles in this router. The scale is shown on the right, and the variance of cycles among the routers is shown below each sub-figure. For the Top and Side placements, it can be seen that there are severe delays that stress CB nodes and/or the nodes surrounding them. This is due to the higher probability that reply packets may encounter each other when CBs are placed at the same row or the same column.

For the Diagonal or Diamond placements, since there are no CBs at the same row/column, the traffic is more balanced, as indicated from the significantly reduced variance values. However, these two placements may cause intersection problems for EIRs due to diagonally neighboring CBs. Consider the two adjacent CB nodes such as the two red-circled nodes in the Diamond (or Diagonal) placement. If the upper CB node has a horizontal (e.g., x-) interposer link connecting an EIR and the lower CB has a vertical (e.g., y+) interposer link connecting another EIR, the two links would inevitably intersect with each other even if both links are only one-hop long. This increases the number of RDLs and yielding cost. In comparison, for the two yellow-circled nodes that are not directly positioned diagonally, intersection can be avoided if the lengths of interposer links are selected carefully. Moreover, neighboring diagonal CBs also increase the contention of injection traffic, which could be mitigated if they are not positioned diagonally.

***N*-Queen Based Placement:** The above analysis prompts us to find a CB placement that minimizes the alignment of CBs in the same row, column or diagonal, so as to reduce traffic contention and be friendly with interposer wiring. These considerations lead us to utilize the N-Queen algorithm that is originally proposed to place N queens on a chessboard where no two queens can capture each other. If used for placing the CBs, such placement ensures that there is only one CB node in a row or a column and there are no CB nodes in diagonal (whether neighboring or not). An example is shown in Figure 3.8 where the placement is very effective, with a variance of only 0.54. This is 35.7% lower than the Diamond placement and 96.7% lower than the Top placement. Moreover, the fact that only one CB node is on any diagonal in the N-Queen placement also decreases the probability of having wire intersection. This increases the flexibility of selecting equivalent injection routers.

Scoring Policy: The N-Queen algorithm does not generate a unique solution. Many N-Queen placements exist, and they may have different impact on traffic congestion. To assess different placements quantitatively, we introduce a concept of *hot zone*. The hot zone of a CB node is defined as the 8 nodes that surround the CB node, as illustrated in Figure 3.9. In particular, the 4 nodes that are connected directly with a CB node are called *Direct Access Zones (DAZs)*. DAZs are very congested as any injected packet is forwarded through DAZs as the first hop. The other 4 nodes at the four corners are called *Corner Access Zones (CAZs)*. Packets have a high probability of being forwarded to CAZs as the second hop. If there is an overlap between the DAZ hot zone of a CB node with the CAZ hot zone another CB node, traffic congestion is greatly exacerbated. Therefore, hot zone overlaps can be used as a metric to assess the quality of N-Queen placements. It is worth pointing out that, in N-Queen placement, it is not possible to have DAZ-DAZ or CAZ-CAZ overlaps, which is another reason why N-Queen is a good placement strategy.

The following scoring policy is introduced. Under a given placement, we calculate a penalty score for each node (tile) in the network, and the summation of the penalty scores of all the nodes is the final penalty score of that placement. Among the four direct neighbors of a node, if m of them are hot zone overlaps, the penalty score of this node is $\sum_1^m$. We use this policy rather than simply adding m points, to reflect the compounded delay by multiple hot zone overlaps. For example, in Figure 3.9, to calculate the penalty score of the red node, we notice that two of its direct neighbors (light yellow nodes) are hot zone overlaps, so the penalty score of the red node is 1+2=3. Note that the node above the red node is a DAZ but is not a hot zone overlap; whereas the two yellow nodes are DAZ-CAZ overlaps. In case of an 8×8 network, there are 92 different N-Queen placements. The one with the lowest score is chosen, as shown in Figure 3.9. For larger networks, a similar procedure is followed to generate a number of N-Queen placements, and the least penalized one is selected.

3.4.3 Selecting EIRs with MCTS

After the CB placement is decided by the above N-Queen and scoring policy, the next step is to select the EIRs. Two observations can be used to simplify the complexity of the selection. First, for a given CB, it is better to distribute EIRs on different directions from the CB, as two EIRs on the same direction would cause contention in that direction. Second, it is better to place EIRs within a few hops from a CB to avoid using long interposer wires and to reduce intersection. However, the design space after these simplifications is still quite large. In the example of an 8×8 network, there are 1.7×10^{10} possible combinations of EIR selections, even if we limit EIRs to be within 3 hops of the corresponding CB node. Therefore, a systematic search approach is needed to identify a good EIR selection. In this work, we develop a search method based on Monte Carlo Tree Search (MCTS).

MCTS is a classic search algorithm in machine learning [11] and has been recently employed to enhance the AI algorithms in the game of Go, Shogi and StarCraft II [47, 48, 4] to search their huge solution spaces. We adopt MCTS due to the inherent similarity between placing EIRs in our problem and placing stones in the game of Go. Other search algorithms might work, such as genetic algorithm (GA) or simulated annealing (SA), but likely at a lower efficiency due to the additional mathematical transformation and less effective problem representation. For example, to utilize GA, a natural representation is to use a 64-bit gene for an 8×8 network, where each bit is either 0 or 1 to indicate if that node is an EIR. This unnecessarily expands the problem space to 2^{64} (or 1.8×10^{19}) and introduces numerous invalid solutions during crossover and mutation operations. Similar issue on problem formulation exists in SA as well.

Our proposed search method follows a typical MCTS, with a few customization and optimizations specific to EIR selection. The search process builds a search tree iteratively. Each iteration consists of four steps, namely selection, expansion, simulation and backpropagation, as shown in Figure 3.10.

(1) *Selection*: Search starts from the root node (e.g., an empty state with no EIR selected) and recursively selects a child node until reaching a current leaf node (e.g., a few EIRs added from previous iterations). This step selects a promising path from the current search tree, so the path can be expanded in the following steps. The selection is based on an

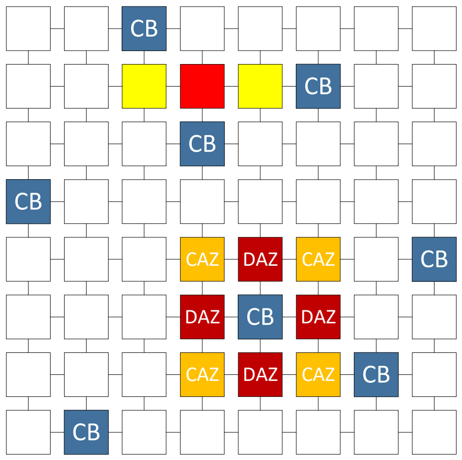


Figure 3.9: N-Queen placement.

Upper Confidence Bound (UCB) formula[2] that balances exploitation, which maximizes the rewards from known nodes, and exploration, which explores unknown nodes for potentially higher rewards [31].

(2) *Expansion*: Assuming the above leaf node is not a terminating node (e.g., last EIR added), this step randomly chooses a possible successive state (e.g., add another EIR or a group of EIRs), attaches to the leaf node, and expands the tree by one level.

(3) *Simulation*: Possible outcomes following the expansion are "simulated" by performing a rollout policy. In case of the Go game, this step means that multiple moves are performed speculatively if the current move is indeed made. Similarly, additional EIRs are selected speculatively (but not actually selected).

(4) *Backpropagation*: An evaluation function estimates the value of the rollout based on a set of defined rules and grading policy. The evaluation score is then backpropagated to all the nodes on the path, starting from the expanded node to the root node. The score is

[2]Defined as $v_i + C \times \sqrt{lnN/n_i}$, where v_i is the estimated value of the chosen child, n_i is its total number of visits, and N is the total number of visits of its parent node. C is a balancing parameter.

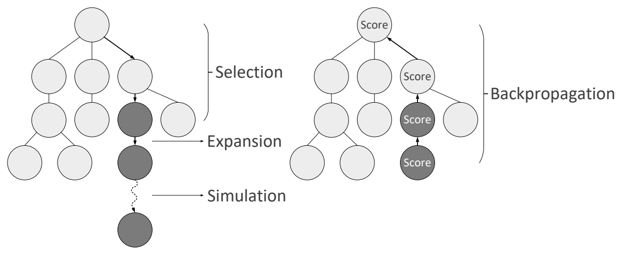


Figure 3.10: Four primary stages of MCTS.

accumulated to the existing scores of those nodes.

At the end of the first iteration, the level-1 child node of the root with the highest accumulative score is considered as near-optimal, and the corresponding EIR selection represented by that node is added to the final EIR selection. This information is carried over to the second iteration as part of the new root state. The second iteration finalizes the EIR selection of a level-2 child (with the highest accumulative score) of the root, and so on so forth, until the EIRs of all the CBs are selected.

The above search process adds EIR one by one. While this is correct, during implementation we observe that the search tree can be as deep as 24 levels for an 8×8 network and even deeper for larger networks. This reduces search efficiency significantly. To address this issue, instead of adding EIR one at a time, we add EIRs group by group. Specifically, each node expansion adds the selection of all the EIRs belonging to a CB node. Therefore, the tree depth equals exactly the number of CB nodes, which is usually limited in a processor.

An important component of MCTS is the evaluation function in backpropagation. We integrate four metrics in the evaluation function to reflect the considerations on EIR efficacy and physical viability: (minimizing) the max of EIR traffic load, average hop count, number of intersection points, and link length. The first metric estimates the total amount of traffic that each EIR needs to handle for all the PE nodes, and the objective is to minimize the maximum one across all the EIRs. This metric helps to balance traffic load among EIRs to avoid hotspots. This is particularly useful as some CB nodes may have fewer EIRs, e.g., due to the consideration of avoiding intersection or simply due to boundary constraints. The second metric uses average hop count to approximate packet latency. The third and fourth metrics consider the physical constraints of RDLs in the interposer. All the four metrics can be easily calculated under a specific EIR selection and CB placement, assuming each PE has relatively similar traffic load. Owing to this, metric calculations can be performed quickly in each backpropagation step to guide the search process; whereas detailed full system simulations with actual workloads are conducted later only for EIR selections that are found to be promising by MCTS. The evaluation function sums the four metrics (after normalization). Lower function values indicate better EIR selections.

We implement the above MCTS model in Python 3.7.3 [43] and execute on an x86_64 Linux server, equipped with an Intel E5-2630v3 32-core processor and 64GB memory. The search process turns out to be quite efficient. For instance, for an 8×8 network, MCTS can stabilize to a near-optimal EIR selection in less than 10 hours by assessing only 0.047% of the entire solution space. Figure 3.11 shows the best design found by MCTS in this case. EIRs with the same color belong to the same group of a CB (in MCTS, we do not allow an EIR to be shared with more than one CB). While the search process is carried out without human intervention, it is interesting to see that MCTS seems to be able to "synthesize" several design attributes of good EIR selections. First, as can be seen, all the EIRs are placed exactly 2 hops away from each CB, despite the up to 3-hop flexibility in the constraint. Closer examination verifies that 2-hop away EIRs bypass both DAZ and CAZ hot zones surrounding the corresponding CB node. Second, intersection is completely avoided, thus requiring minimally only one RDL in the silicon interposer. Furthermore, interposer links with 2-hop length can be fit into one clock cycle, thus avoiding the use of repeaters and active interposers. Note that those 2-hop links are routed in the interposer, so they do not interfere with the placement of regular links in the processor die.

As the network size increases, one might think that EIRs located multiple hops away from CBs are more efficient. However, the 2-hop away EIRs actually work very well for larger networks. First, the number of routers to distribute the injected packets increases geometrically as the packets are forwarded further into a network (e.g., 4 routers after the first hop, 16 routers after the second hop, etc.). Thus, for larger networks, the bottleneck is still at the region close to injection points. Second, we have observed that, the contention delay from injection quickly drops after one or two hops after the injection points, regardless of the network sizes. Therefore, using interposer links to bypass the first two hops are sufficient to avoid injection contention even in large networks. For extremely large networks, if the need arises, it is possible to use slightly longer interposer links (e.g., 3-hop links). Intersection can still be avoided as those networks provide more space to place non-intersected links. However, as we show later in the Evaluation section, two hops are more than enough even for 16×16.

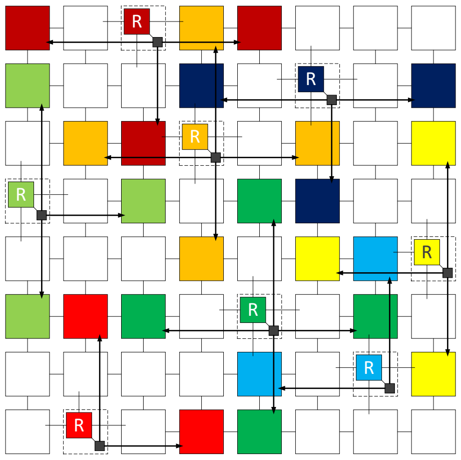


Figure 3.11: EquiNox in case of an 8x8 NoC. Blocks with the same color belong to the same CB group.

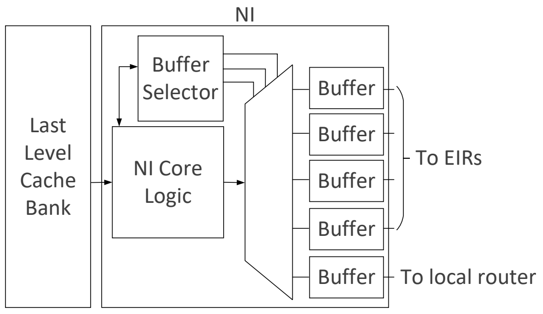


Figure 3.12: CB-connected NI architecture in EquiNox.

3.4.4 Modifications to Microarchitecture

To implement EquiNox, some modifications are needed to the architecture. In the original NI structure, an injected packet from the last-level cache bank is serialized by the NI core logic and then stored in a packet injection buffer. This buffer directly connects to the local CB-connected router. The size of the buffer is usually a few packets, although only one flit is sent to the connected router in a given cycle. With the use of EIRs, the CB-connected NI needs to connect to multiple equivalent injection routers. Consequently, the NI structure needs to be modified, as depicted in Figure 3.12. The main change is that the injection buffer is split to five single-packet buffers, where four of them are connected to the four EIRs that are two hops away through interposer, and the remaining one is connected to the original local router. To reduce design and verification effort, all the CB-connected NIs use this architecture, even though some ports can be left idle if there are fewer EIRs connected. Later in the evaluation section, we configure the original NI to have one packet-sized injection buffer and the modified NI to have five packet-sized injection buffers to appropriately account for the overhead. In addition, a de-multiplexer is inserted to switch injected packets to different injection buffers, and a buffer selection signal is generated by the *Buffer Selector* when the NI core logic is processing a packet.

To avoid detouring, injected packets are only allowed to use the EIRs on the shortest paths (or the local router which is also on the shortest path), even though other EIRs might have less traffic. This is fulfilled by the Buffer Selector. Specifically, the relative position of a packet's destination (x_d, y_d) with regard to its source (x_s, y_s) is first generated. Based on the relative position $(\Delta x, \Delta y)$, there are 8 possible relative directions, of which 4 are right on the axis (either Δx or Δy is 0) and 4 are inside the four quadrants. If a destination is on an axis, there is one and only one EIR that is on the shortest path, and that EIR is selected. If the destination is inside one of the four quadrants, there are up to two shortest-path EIRs exist (except for boundary cases). In this scenario, round-robin is used to select the EIR. In either case, if the buffer that connects to the to-be selected EIR(s) is not available, the packet is injected into the local CB-connected router. Detailed selection policy is presented in Buffer Selection 1. Note that it is not possible for both Δx and Δy to be 0 as MC nodes do not send packets to themselves.

While not shown in Figure 3.12, the EIR on the receiving side needs to accept injected

Buffer Selection 1: Buffer Decision Policy

```
if $\Delta x == 0$ or $\Delta y == 0$ then
    if the buffer is available then
        Inject to the buffer associated with the node on destination direction
    else if local buffer is available then
        Inject to the local buffer
    else
        Retry next cycle
else if $\Delta x\ != 0$ and $\Delta y\ != 0$ then
    if 2 buffers are available then
        Choose 1 buffer in round-robin fashion
    else if 1 buffer is available then
        Inject the packet to the buffer
    else if local buffer is available then
        Inject to the local buffer
    else
        Retry next cycle
else
    Error
```

packets from the split NI injection buffer. Therefore, one input port is added to the EIR. Note that this is needed only for routers that are selected as EIRs and only for the reply network, while routers in the request network is unchanged. Some EIRs are located on the boundary, so an unused input port might already be available, if the boundary routers use the same router template as the non-boundary ones (to reduce verification cost). Either way, the amortized overhead is only a few percent and much better than existing alternatives to mitigate the injection bottleneck, as presented in Section 7.

Deadlock freedom is a related critical issue that should be discussed. First of all, EIRs do not affect existing virtual channel (VC) allocation. Packets that are injected from an NI to an EIR use the same VC allocation policy as if the packets are injected to the local router. Also, without detouring, the addition of EIRs do not change the routing policy in the original network. Therefore, if the channel dependence graph of the original network is acyclic, the EIRs do not introduce new cycles and are free from routing-induced deadlocks. Regarding protocol-introduced deadlock, since the request and reply packets are routed through two physical networks, there is no dependence at the endpoints and no protocol-introduced deadlock either. Thus, EquiNox is deadlock-free.

To summarize, with the combination of a N-Queen based CB placement, a set of carefully placed equivalent injection routers determined by MCTS, and the needed microarchitecture modifications to enable correct and deadlock-free operations, the proposed Equnix effectively removes the injection bottleneck and utilizes interposer wiring resources. In the following sections, EquiNox is evaluated quantitatively.

3.5 Evaluation Methodology of EquiNox

The proposed EquiNox design is evaluated using a combination of architecture and RTL level simulators. Due to the lack of a comprehensive cycle-accurate simulator that models HBM and interposer, we have developed an integrated simulation environment by combining and heavily modifying BookSim 2.0 [22], GPGPUSim 3.2.3 [7] and Ramulator [30]. The NoC simulation is performed by the cycle-accurate simulator BookSim 2.0 that includes all the on-chip network resources (e.g., links, routers, network interfaces). We integrate Ramulator with GPGPUSim to enable HBM simulation. The simulated HBM contains 8 chips, each having 4 stacks. There are 64 TSV IOs per channel and 16 channels

Table 3.1: Key Parameters in Simulation.

Parameter	Value
Network size	8x8, 12x12, 16x16
Network routing	Minimum adaptive
Virtual channel	2/port, 1 pkt/VC
Allocator	Separable input first
PE frequency	1126MHz
Shared memory / PE	48KB
L1 cache / PE	16KB
L2 cache (LLC) per bank	2MB
# of LLC banks	8
HBM bandwidth	256GB/s per stack
# of Memory dies / stack	4
Memory controllers	8, FR-FCFS

per chip, totaling 1024 IOs for each chip. The physical layer PHY that has 8 channels and locates on top of each memory stack [33] is also simulated as the interface between HBM and memory controllers. The area and power consumption of NoCs are based on DSENT [49] which is extensible to simulate novel components of NoCs. Following the methodology of prior works on interposer [21, 18, 57], we modify and extend DSENT to model interposer links. To evaluate area overhead more accurately, we use Verilog HDL to implement the RTL of new components in EquiNox that are not modeled in DSENT. A standard VLSI design flow is followed that employs Design Compiler for logic synthesis using 28*nm* process technology [56]. Table 3.1 lists the key configuration parameters. A 8×8 Mesh NoC is assumed for the main simulations, and 12×12 and 16×16 NoCs are also simulated for scalability study. A wide range of 29 benchmarks from Rodinia [13] and Nvidia CUDA SDK [40] are executed to evaluate the performance of different proposed schemes.

We compare the following seven schemes. In particular, schemes (1), (2) and (3) are based on the single network type where the request and reply networks share the same physical network. Schemes (4), (5), (6) and (7) are based on the separate network type where the request and reply networks have separate physical networks. In both types, we include a baseline, one or two recent but conventional (no interposer) schemes, and an interposer-based design. Below are the details.

(1) SingleBase: baseline for the single network type with Diamond placement and minimal adaptive routing.

(2) VC-Mono [19]: a recent scheme that increases network throughput by allocating all the VCs (monopolization) to either request or reply traffic if only one of them is present.

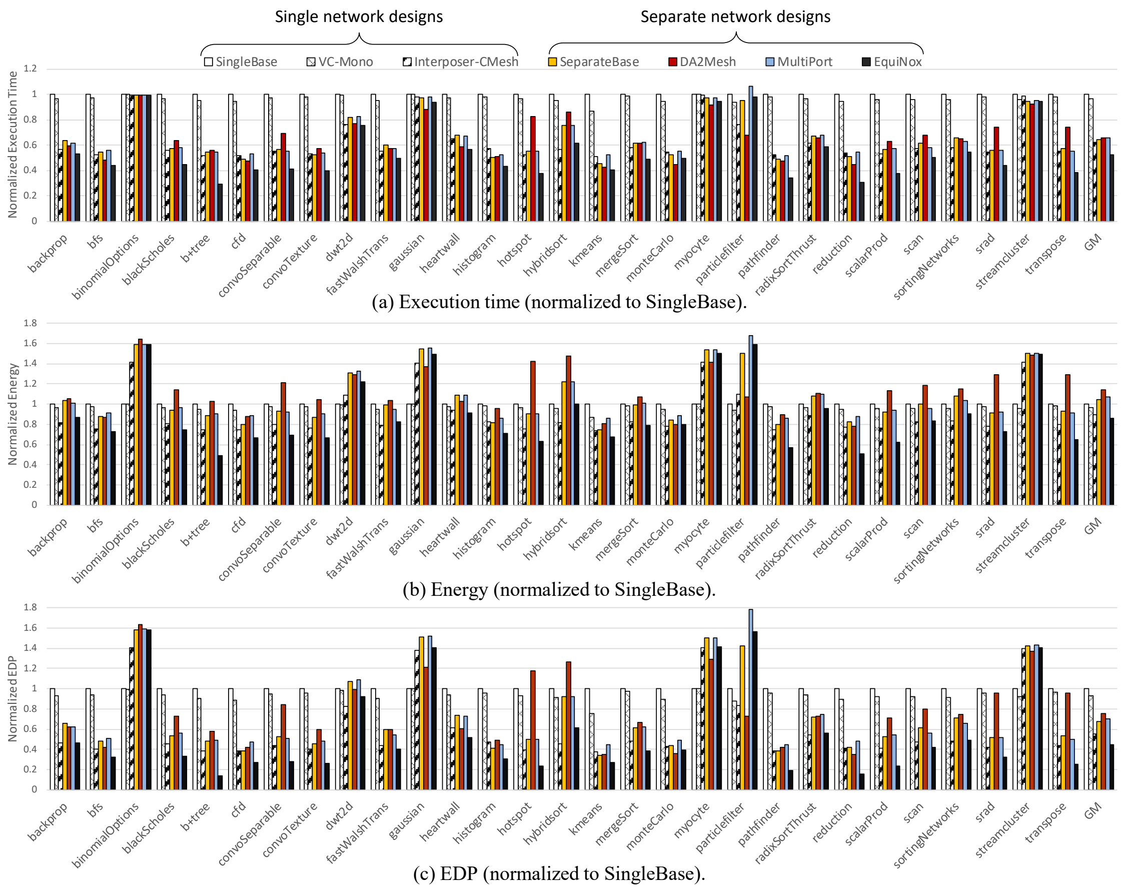
(3) Interposer-CMesh [21]: a state-of-the-art scheme for many-core CPU NoCs that introduces an additional CMesh network whose links are in the interposer.

(4) SeparateBase: baseline for the separate network type with Diamond CB placement and minimal adaptive routing.

(5) DA2Mesh [27]: a recent scheme that splits the reply network into eight narrow subnets with 1/8 flit size; the network frequency is set to 2.5X of the baseline in [27].

(6) MultiPort [6]: a scheme that uses multiple injection (and ejections) ports for all CB-connected routers to mitigate the reply injection bottleneck.

(7) EquiNox: the proposed scheme with N-Queen placement and MCTS-selected EIRs, as described in Chapter 5.



(a) Execution time (normalized to SingleBase).

(b) Energy (normalized to SingleBase).

(c) EDP (normalized to SingleBase).

Figure 3.13: Comparison of Execution time, energy and energy-delay product (EDP).

3.6 Results and Analysis of EquiNox

This section presents the detailed evaluation results. As single network schemes and separate network schemes have very different performance-energy characteristics, we thereby juxtapose execution time and energy consumption with ener-gy-delay product (EDP) to reflect the trade-off more accurately.

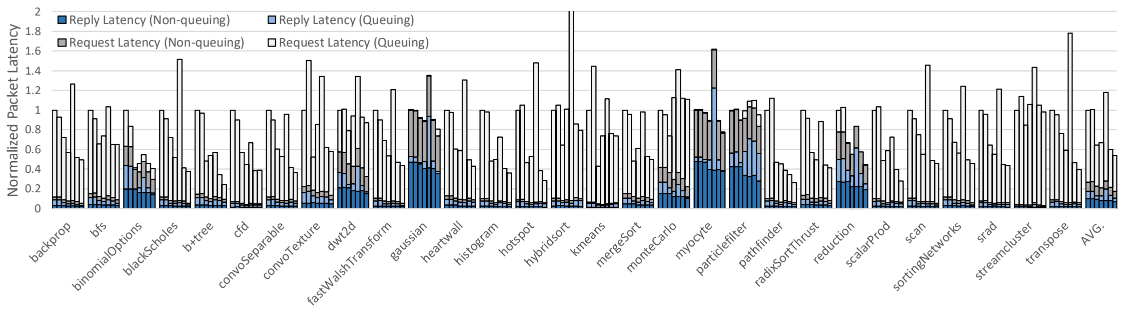


Figure 3.14: Normalized packet Latency (the bars from left to right are: *SingleBase*, *VC-Mono*, *Interposer-CMesh*, *SeparateBase*, *DA2Mesh*, *MultiPort*, and *EquiNox*).

3.6.1 Effect on Performance

Figure 8(a) plots the execution time for the seven schemes, normalized to SingleBase. VC-Mono reduces the execution time by 3.6% on average, although large reduction is observed in some benchmarks (e.g., 13.1% in kmeans) because of better utilization of virtual channels via monopolization. However, these two conventional single network schemes (SingleBase and VC-Mono) do not provide additional bandwidth to mitigate the reply injection bottleneck and thus have lower performance than other schemes. The third single network scheme, Interposer-CMesh, achieves 37.9% execution time reduction compared with SingleBase. This is mainly due to the use of an extra network formed by interposer links in the RDLs. The separate network schemes perform noticeably better than the single network schemes in general, as the separate network schemes provide a dedicated network for the reply traffic, thus having more injection bandwidth. DA2Mesh obtains sizable reduction in execution time for heartwall, kmeans, monteCarlo and particlefilter, whereas MultiPort has larger improvement for fastWalshTrans, scan and sortingNetworks. However, the two schemes do not seem to perform much better than the SeparateBase when averaged over all the benchmarks. For DA2Mesh, this is due to the high packet serialization latency from its narrow subnetwork links[3]. For MultiPort, although multiple ports widens the injection bandwidth at CB-connected routers, the injected traffic cannot be quickly transferred out, due to the traffic contention in the hot zone surrounding CB nodes. It is interesting to see that the three conventional separate network schemes (SeparateBase, DA2Mesh and MultiPort) have slightly lower performance than Interposer-CMesh. This shows that it is beneficial to exploit interposer links. Finally, the proposed EquiNox reduces execution time by 47.7% compared with SingleBase and by 23.5% compared with SeperateBase, which is the largest reduction among all the schemes. In particular, EquiNox performs much better than Interposer-CMesh as EquiNox specifically addresses the injection bottleneck that is unique to throughput processors.

3.6.2 Effect on Energy

Figure 8(b) compares NoC energy consumption. As can be seen, the three conventional separate network schemes (SeparateBase, DA2Mesh and MultiPort) have the highest energy due to the overhead of two physical networks. While being a separate network scheme, EquiNox has low energy consumption because of small power overhead and large reduction in execution time. On average, EquiNox achieves 15.0% less NoC energy than SingleBase and 18.9% less than SeparateBase.

[3]The improvement of DA2Mesh is smaller here than what is reported in [27] due to differences in benchmarks and settings. As verification, our implemented DA2Mesh performs similarly as in the original paper if those factors are the same.

3.6.3 Effect on Energy-Delay Product

Figure 8(c) examines the normalized EDP of the compared schemes. The conventional single network schemes (SingleBase and VC-Mono) have higher EDP than the separate network schemes, as single network schemes have higher execution time but similar energy consumption. However, Interposer-CMesh that exploits the interposer layer is able to reduce the EDP by 44.7% compared with SingleBase, and achieves 26.4% and 21.0% less EDP when compared with DA2Mesh and MultiPort, respectively. The results indicate that the interposer-based designs are useful in exploiting the interposer opportunity to achieve better performance-energy trade-off. The proposed EquiNox is able to reduce EDP by 55.0% when compared with SingleBase and by 32.8% compared with SeparateBase. These large improvement further supports the observation that interposer-based designs are promising for high-performance and energy-efficiency. In addition, the EDP of EquiNox is 18.2% lower than Interposer-CMesh, if compared relatively. This demonstrates the need for optimizing the use of interpose links for specific traffic behaviors in throughput processors.

3.6.4 Reduction of Packet Latency

To gain more insights on why EquiNox is able to achieve a large performance improvement, Figure 3.14 plots the NoC packet latency, breaking down into queuing and non-queuing parts for both request and reply latency. The scheme bars follow the same order as Figure 9 from left to right. Because DA2Mesh has a different NoC frequency, all the results are converted to nanosecond (*ns*) and then normalized to SingleBase to provide an accurate and fair comparison. Due to normalization, for benchmarks such as gaussian and myocyte, the bars do not indicate that their non-queuing latency is high; they just mean that most of the latency is from the non-queuing part.

On average, the figure shows that request latency is considerably higher than reply latency. This might be non-intuitive since the bottleneck is at the reply injection point. However, this is correct because the congestion at reply injection creates a backpressure that is propagated to the request network. Analogous to the classic parking lot problem with a congested exit point, the cars that are the farthest from the exit experience the longest waiting time. Similarly, packets in the request network experience longer NoC latency even though the actual congestion occurs in the reply injection. This trend is consistent with prior observations [6, 19, 27, 60].

Overall, the single network schemes have relatively higher packet latency. This is expected due to the traffic contention between mixed request and heavy reply traffic. With interposer links, Interposer-CMesh reduces packet latency by 35.6%. SeparateBase and MultiPort have similar reduction in packet latency, with 33.1% and 40.2% on average, respectively. The highest average packet latency is observed in DA2Mesh. Further inspection shows that this is mainly caused by a much higher serialization latency. For the proposed EquiNox, it has the lowest reply packet latency as the use of EIRs addresses the bottleneck at the reply injection. It can be seen that the queuing part of the request latency is reduced significantly in EquiNox, due to the backpressure explained above. Compared with SingleBase, EquiNox greatly reduces the request, reply and total packet latency by 44.6%, 40.6% and 45.8%, respectively.

3.6.5 NoC Area

We have also assessed the area cost of various schemes, plotted in Figure 3.15. As expected, conventional single network schemes have lower area than separate network schemes. The only exception is Interposer-CMesh, which has an extra concentrated mesh network with 16 routers (links are in the interposer but not the routers). Moreover, these routers have 2x more ports than a basic router because of the need to handle both concentrated traffic and inherent CMesh traffic. This contributes to a higher NoC area for Interposer-CMesh. On the separate network schemes side, DA2Mesh has lower area due to the narrower and simpler routers, whereas MultiPort and EquiNox have higher area than SeparateBase due to the use of extra ports. In particular, with the additional components such as added buffers in NIs and added input ports in EIRs, the proposed EquiNox consumes 4.6% more die area than SeparateBase.

3.6.6 Comparison of μbumps Area

This subsection compares the μbumps area for Inter-poser-CMesh and EquiNox. To support the additional concentrated mesh network, Interposer-CMesh needs 128 uni-directional links between the processor die and interposer, with 256 bits for each link. This leads to a total of 32,768 μbumps for physical and electrical connectivity in Interposer-CMesh. In contrast, the proposed EquiNox has 24 uni-directional 128-bit links and two μbumps for each wire (from the process die to interposer and back to the processor die), resulting in 6,144 μbumps. The μbumps overhead is significantly reduced in EquiNox by 81.25%. The large saving comes from the fact that EquiNox utilizes the M2F2M

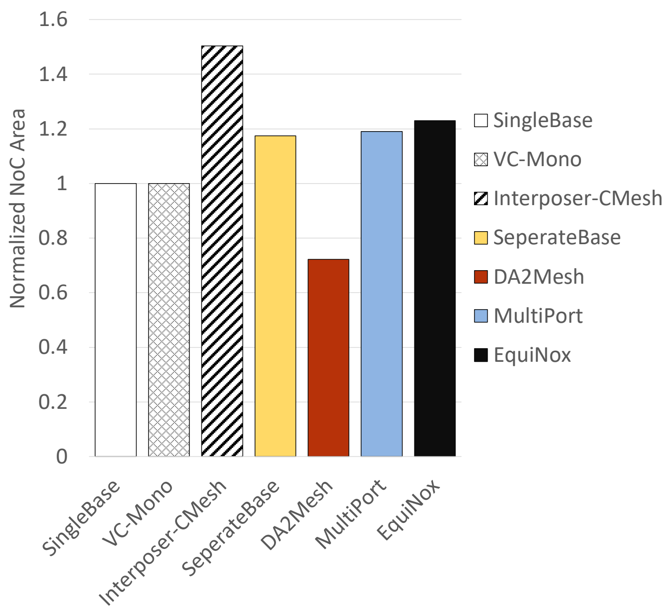


Figure 3.15: NoC area comparison.

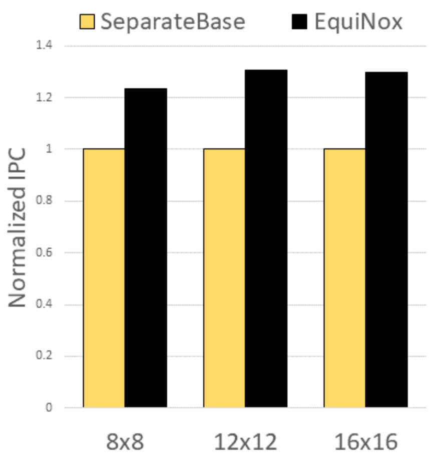


Figure 3.16: Scalability.

traffic pattern and strategically places only a few EIRs. Note that neither schemes has intersection of interposer links, so one RDL is sufficient in both schemes.

3.6.7 Scalability

As analyzed in Section 4.3, the proposed scheme is expected to work well with larger network sizes. To verify this, we follow the same design flow of N-Queen and MCTS for 8×8 to generate EquiNox versions for 12×12 and 16×16 networks. As compared in Figure 3.16, the performance improvement (average IPC) is 1.31x in 12×12, and 1.30x in 16×16, both of which are greater than the 1.23x in 8×8. Larger networks may have more serious injection bottleneck issue, so the impact of EquiNox becomes greater. These results demonstrate the excellent scalability of EquiNox.

3.6.8 Discussion

One potential issue is how to deal with the number of CBs that is greater than N. Apparently, there will be more than one CBs on at least one row, column or diagonal. Due to space constraint, we state here without providing proof that, if the number of CBs is greater than N in a $N * N$ layout, placing CBs following the knight-move shape in chess can lead to the lowest occurrence where two CBs are on the same row, column or diagonal. The scoring policy is still applicable except that hot zone overlaps may be between DAZ-DAZ and CAZ-CAZ. The remaining steps are the same as Section 4.2.

If the number of CBs is less than N, the redundant CBs from the N-Queen solution can be randomly deleted, and the scoring policy can be used to select the (near) optimal one.

Chapter 4: Machine Learning-based Responsiveness and Battery Life Prediction

4.1 Background and Motivation

PnP has become a chief area of competition. Devices including laptops, notebooks, tablets, 2-in-1 compute devices, smartphones are primarily operated without an AC cord plugged in so battery life becomes particularly important for connected scenarios, gaming scenarios, video playback and related use cases. Meanwhile, the system performance is also a major customer consideration when selecting a new device. As the techniques that enhance the system performance usually consume higher energy which reduces the battery life, the trade-off between performance and power is difficult to be balanced. That is, the PnP requirements have become the most important features when designing and optimizing a computing system.

Original equipment manufacturers (OEMs) need the PnP values for newly designed systems to check whether they meet the PnP requirements. However, industry companies such as Intel often test PnP values by manually swapping hardware components on a system. Such processes costs a significant amount of time, and the OEMs have to wait for a response from Intel. More importantly, manual hardware swapping requires human resources to supervise the process and to hand tune the system hardware configurations to meet the system PnP requirements. For OEMs, it is imperative to have a fast and accurate method to speed up the hardware tuning process. For an industrial company like Intel, it is imperative to reduce the number of experts on such repetitive and tedious tasks and move them to more creative positions. Therefore, a solution is urgently needed to predict the PnP values for newly designed systems, so that the need of OEMs and Intel can be satisfied at the same time.

Intel has already developed a tool called Docea to partially resolve the issue. However, Docea is a pure computational model which is based on the default and constant parameters and weights. The major issue with Docea is that the calculation accuracy is less than satisfactory. This is because the internal calculations are all based on the parameters and weights, while in the real world, the PnP-related parameters and weights are dynamic in different systems. Therefore, the constant parameters and weights cannot reveal the real hardware behavior and produce low calculation accuracy.

In this research, we propose a natural language processing (NLP) based solution which integrates several models and provides relatively high PnP prediction accuracy. The solution firstly learns the inter- and intra-relationship among the components and system features and transfers the string data in the dataset to floating point word vectors by using an enhanced Continuous Bag of Words (CBOW) model. This model resolves the input issue of using string data and is also helpful for the purposes of dataset analysis. We also propose a model, representing the hardware, that utilizes the component specs to predict the word vectors for future unseen hardware models of the existing components. Finally, a newly designed system can be represented by combining the transformed word vectors from the original dataset and the predicted word vectors from the model. The word vectors of this system are then fed into a PnP prediction model to get the predicted

PnP values for the system.

To summarize, in this work, we design and implement a solution which takes measured system components and features as inputs and predicts PnP values for the systems with unseen component models. This solution contains three different models. Firstly, The enhanced CBOW model learns the component intra- and inter-relationships and transforms component string data to floating point vectors. Secondly, The hardware representing model predicts the word vectors for future unseen hardware models. Thirdly, the PnP prediction model infers the PnP values for any input systems.

4.2 Problem Statement

The specific problem we try to resolve can be stated as follows: suppose a bill-of-material (BOM) or an anchor system along with its response time and battery life values are measured and provided, can the solution predict the response time and battery life for derivative components, with a variety of specs, which may appear in future design? We define the anchor system as the systems which were already tested and proved to meet all Intel Athena Project requirements. The requirements are the threshold of key experience indicators (KEIs). If all measured values of a system meet the threshold of the corresponding KEIs, then the system is considered an anchor system.

A specific example of this problem is, if we have the PnP values of a virtual system which is composed of an i5 CPU, a 16GB memory and a 256GB SSD, is it possible for our solution to predict the PnP values for a system with an i7 CPU, a 32GB memory and 512GB SSD? Suppose the prediction is possible, then OEMs have a reliable solution to assert a small range of configuration space, and industrial companies like Intel can also reduce the human experts on configuration hand tuning tasks.

4.3 Dataset Analysis

Our integrated dataset has 15 system features for each system including power related parameters and component models. The dataset contains 215 unique systems, where about 1/4 are anchor systems and the rest are derivative systems. As mentioned previously, the anchor systems meet all Intel Athena Project requirements. 28 out of 35 KEIs are response time KEIs and the rest 7 are battery life KEIs. A KEI is an operation during system runtime, such as opening Microsoft Word or streaming an online video. Due to the sensitivity of this information, we are not going to discuss the details of our dataset.

4.4 Motivation of NLP-based Solution

As the PnP values are closely related to the hardware components and system paramters, an accurate prediction solution should learn the inter- and intra-relationship among the input features of a configuration and among all measured configurations. Interestingly, the input dataset can be considered as a special 'language'. In such language, each configuration is a sentence, a system parameter or a component is an element in a sentence and each specific value or model of a system parameter or a component is a word in the vocabulary of this language. The inter- and intra-relationship thus can be revealed by such a language. Different components or system parameters may have

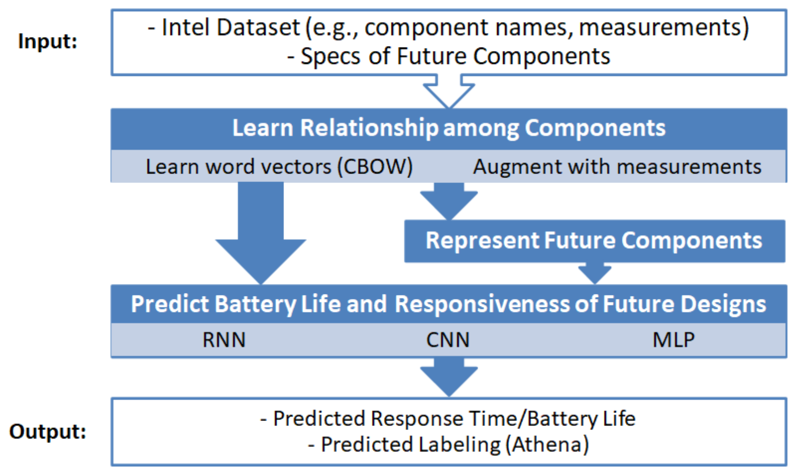

Input:
- Intel Dataset (e.g., component names, measurements)
- Specs of Future Components
Learn Relationship among Components
Learn word vectors (CBOW)
Augment with measurements
Represent Future Components
Predict Battery Life and Responsiveness of Future Designs
RNN
CNN
MLP
Output:
- Predicted Response Time/Battery Life
- Predicted Labeling (Athena)

Figure 4.1: The prediction flow of the proposed NLP-based solution.

inter-relationship, this can be analogous to different elements of a sentence, such as subject-predicate-object. For example, SSD-memory-CPU may be closely related to the PnP values. The intra-relationship of different models in a component can be analogous to many words of the same element in a sentence in English, such as you/we/they can all be the subject in a sentence. Taking SSD as an example, Samsung/Micron/SK Hynix can all be the vendor of SSD and could related to the PnP results. Such observation prompts us to use NLP-like solution to explore the inter- and intra-relationship in the dataset and predict the PnP values.

4.5 Overview of NLP-based Solution

To clarify our proposed solution, we demonstrate our overview flow in Figure 4.1. The inputs of our solution are the measured dataset and component specs from Intel, and the outputs are the predicted PnP values and the predicted pass/fail labels for each system.

The CBOW algorithm with carefully designed enhancements is employed in our first model. The enhanced CBOW model transforms string data such as component names or system features in the dataset to the floating point vectors which are so called word vectors that contain the information of the inter- and intra-relationship among the components. Then, in the second step, the component specs and the word vectors of the existing components are fed into the hardware representing model to predict the word vectors for future unseen components. For example, suppose the word vector of a memory model with 8GB and 1333GHz is generated by the enhanced CBOW model in the first step. The word vector of an unseen memory model with 16GB and 2666GHz can then be easily predicted by the hardware representing model. In the third step, we take the word vectors of a derivative system with some unseen models from the first two steps, then utilize a ML model to predict the PnP values and the system label information. We use different models such as MLP, RNN and CNN in this step to study the impact of different models.

4.6 Enhanced CBOW Model

The majority of cells in the dataset are string data, such as component model names. It is impossible for the string data to be the inputs for any machine learning model. Therefore, the string data should be transformed to floating point vectors which can be accepted by machine learning models. In this section we introduce our proposed model to achieve the necessary transformation.

One-hot transformation is a conventional and efficient solution to transform strings to floating point vectors. The size of one-hot vectors is exactly equal to the total number of words that should be transformed. Each one-hot vector only contains 0s and a single 1 to differentiate itself from all the others. The one-hot vectors are easy to generate but have some crucial defects. First of all, there are a significant amount of 0s in a one-hot vector, as the size of each vector is the total number of input words. These 0s notably increase the training complexity for any machine learning model as most of the weights cannot be updated in each training epoch, and the 0s also could potentially decrease the prediction accuracy. Another defect of one-hot transformation is that the meaning of each word cannot be represented accurately. The one-hot transformation only ensures that there is a single 1 in each vector and any two vectors do not have the same position of their 1. Therefore, the distance between any nearby two words is constant. That is, a machine learning model would consider the nearby words to be related and have about the same distance for every pair of nearby words. However, when considering the meaning of these words, it is impossible for them to be the same distance. In reality, the relationship among the input words is more like a complicated network while the one-hot vectors can only reveal linear relationship. In conclusion, one-hot transformation is only suitable for those datasets that do not care about the distance between the input words. In our project, the input words are actually the hardware model names which are related to the PnP values. That is, the distance and meaning of each word is important and should be carefully considered.

To compress the one-hot vectors and derive the meaningful vectors, we introduce the Continuous Bag of Words (CBOW) model. The CBOW model takes the one-hot vectors of the context of a center word as inputs, and predicts the center word. For example, suppose we have a sentence "I have a computer", then I/have/a/computer can all be the

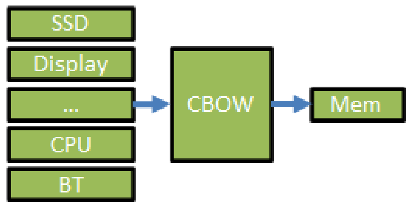


Figure 4.2: CBOW on Intel dataset.

center word, and the rest of the words in the sentence are the context.

A typical CBOW model has two layers, suppose there are in total W words in a vocabulary and the pre-determined word vector size is s. Then, the first layer of CBOW has s hidden units and the second layer has W output units. Thus, the size of the weight matrix M_1 of the first layer is $W \times s$ where each row vector with size $1 \times s$ is the word vector we look for. At the start of the CBOW training process, each word is labeled with a unique one-hot vector with size $1 \times W$ and a single 1. The one-hot vectors of the context words are summed together to form a new vector as the input, so the input vector size is also $1 \times W$, but contains N 1s where N is the number of context words. The output size is $W \times 1$ where each cell is the probability that a word is the center word. Then, we select the word with maximum probability as the center word. In the training process, We employ negative log-likelihood loss in the output layer to perform the backpropagation. After training the CBOW model with the context-center-word pairs, the model learns the potential relationship for each word as the center word with its context words. To extract the word vector for a word, the one-hot vector of this word is fed into the CBOW model, and the word vector is the only activated row in M_1 as there is always a single 1 in each one-hot vector. More specifically, the one-hot vector X with size $1 \times W$ is multiplied by M_1, where the size is $W \times s$, and the result is a vector whose size is $1 \times s$ and is exactly

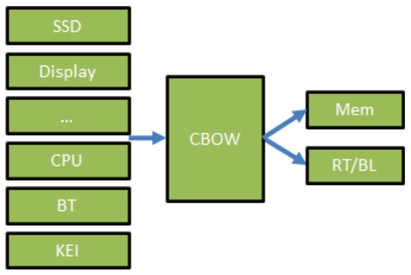


Figure 4.3: The enhanced CBOW model.

the word vector for this word.

Word vectors implicitly contain the context information. In our dataset, as shown in Figure 4.2 each parameter and component model can be the center word and the rest of the system features are the context. That is, the word vector for each system parameter or component model contain the information of which “role” that the parameter/model plays in a configuration.

Nevertheless, the ordinary CBOW model is designed to catch the context relationship in natural languages. Besides the context relationship, the configuration behaviour (*i.e.* measured values of KEIs) is another piece of information that the word vectors should include. The configuration behaviour is a standard score for evaluating a configuration, and such information helps the following models that use the generated word vectors predict PnP values more accurately. To integrate the configuration behaviour, we add the KEI index as one of the inputs and the measured value of the KEI as an output. Figure 4.3 depicts the enhanced CBOW model for our dataset. There are two outputs, the center word and the predicted PnP value under the current configuration.

To obtain the best word vectors for each system parameter or component model, we train the enhanced CBOW multiple times with different training parameters, such

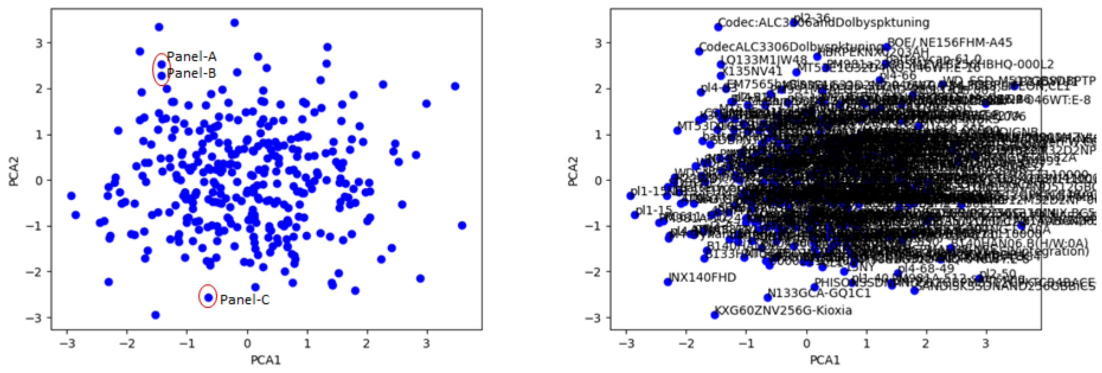


Figure 4.4: The unlabeled (left) and labeled (right) figures of generated word vectors after applying PCA algorithm.

as learning rates or the weights of the two outputs. This is because the two outputs use different loss functions. The center word output uses classification loss (*i.e.* cross entropy loss) and the PnP value output uses regression loss (*i.e.* mean square error loss). In our evaluation, the two losses have different gradient step scales, the learning rates of the center word output and the PnP value output are around 1×10^{-3} and 1×10^{-6}, respectively. Moreover, the two losses have opposite gradient descent directions, that is, the decreasing of one loss leads to the increasing of the other loss and vice versa. Therefore, the enhanced CBOW model is tuned and trained multiple times to ensure the best word vectors are collected.

After numerous tests, we see that the word vector size 1×12 provides the best performance in our case. There are 345 words in the dataset which come from 15 different features (*e.g.* SSD, memory, display panel, power level, CPU *etc.*). To evaluate the correctness and accuracy of the enhanced CBOW model, we utilize principle component analysis (PCA) algorithm to map the 1×12 high dimension word vectors to 2-dimension

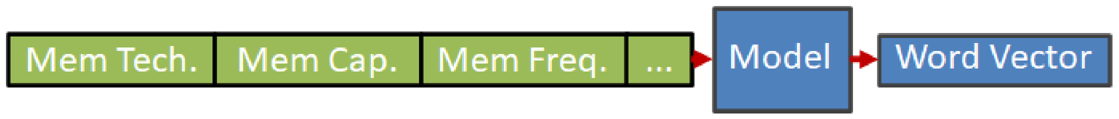


Figure 4.5: The inputs and output of the proposed hardware representing model.

visualized space as shown in Figure 4.4. The neighboring points on the figures meaning that they have relatively similar impact to the PnP values. We circle some models in the left figure of Figure 4.4 as examples to demonstrate our generated word vectors. There are three example panel models named A, B and C. The Panel-A and Panel-B both have 13 inch sized panel and 1920×1080 resolution the only difference is their brightness, so the two word vectors stay near by each other. Panel-C also has 13 inch sized panel but its resolution is 2560×1600 with higher brightness than the other two. That is, Panel-C locates faraway from the other two example panels. The other data points on the figures exhibit the same location pattern.

The enhanced CBOW model generates word vectors for the system features in the previously collected and measured configurations. However, for future unseen system parameters or component models, there should be another algorithm to predict the word vectors.

4.7 Hardware Representing Model

The word vectors of unseen component models cannot be generated by the enhanced CBOW model as there is no measured PnP values for the configurations that contain the unseen models. To predict the word vectors for the unseen models, we propose a machine learning based approach.

For any hardware models, there are several technical specs to differentiate a given model from the others. For example, a memory model can be differentiated by memory capacity, memory technique (*e.g.* DDR2/3/4, LPDDR4 *etc.*), memory main frequency and burst frequency, memory read/write speed, memory read/write timing *etc*. This observation enlightens us to the fact that the specs can also be the input features of a ML model to predict the word vectors as shown in Figure 4.5. The training dataset includes the hardware models that appear in the configurations which are measured. After multiple phases of testing and tuning on different ML models, we introduce a Multi-Layer Perceptron (MLP) model with one hidden layer and 128 hidden units in this case as the hardware representing model. However, this model is proposed and tuned for the current dataset. If there are more hardware models or complicated hardware specs which are introduced into the dataset in the future, this model can also be replaced by any complex machine learning model that supports regression loss.

With this hardware representing model, the word vectors of future unseen hardware models can be predicted. Therefore, the derivative systems are able to be represented by word vectors. In the final step, we propose a PnP prediction model to take the word vectors of a system and infer the PnP values.

4.8 PnP Prediction Model

There are two types of system features in a derivative system, the unseen and the existing features. Each feature in a system should be transformed to a corresponding word vector. The word vectors of existing features are retrieved from the enhanced CBOW model, and

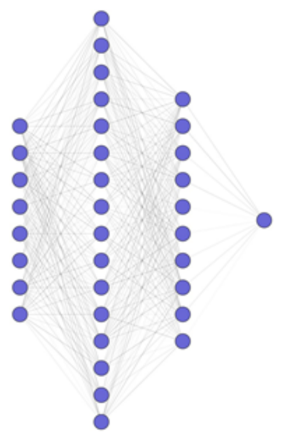


Figure 4.6: The example of the proposed MLP structure.

the word vectors of unseen features are predicted by our hardware representing model. The word vectors of a system are then fed into a PnP prediction model to perform PnP value prediction. There are three potential candidates that we evaluated for the PnP prediction model, which are multi-layer perceptron (MLP), recurrent neural network (RNN) and convolution neural network (CNN).

4.8.1 Multi-Layer Perceptron (MLP)

MLP is a classic machine learning model which has proven to be efficient when applied to datasets with divisible data distribution. As there is no prior information for the dataset distribution in our case, we introduce MLP to provide a baseline prediction accuracy. As shown in Figure 4.6, our best performance MLP has an input layer, an output layer and two hidden layers, with the first hidden layer having 128 hidden units and the second layer containing 64 hidden units. The input feature size equals $S_{word\_vector\_size} \times N_{system\_features}$ where $S_{word\_vector\_size}$ is the word vector size of our

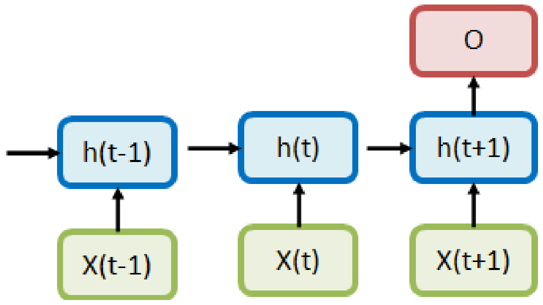


Figure 4.7: A typical RNN structure.

vocabulary (*i.e.* 1×12) and $N_{system\_features}$ is the total number of system features in a configuration, *i.e.* 15 in the dataset. That is, the size of the input features is $12 \times 15 = 180$. The output size is always 1 as the target is a PnP value. For performance KEIs, the value is response time, and for power KEIs, the value is battery life.

4.8.2 Recurrent Neural Network (RNN)

RNNs are suitable for exploring the temporal locality in sequence data. The Figure 4.7 depicts a typical RNN, it has an input layer (X_{t-1} to X_{t+1}) which takes the input features sequentially, a hidden layer (h_{t-1} to h_{t+1}) which takes the current input feature and the last hidden state, and an output layer (O). The h_t is calculated by an equation: $h_t = f_h(W_h X_t + U_h h_{t-1} + b_h)$. The current input X_t and the last hidden state h_{t-1} is included in the equation, so the h_t combines the memory from the previous hidden layers and the current input. Thus, RNNs catch the temporal locality by recurrently using the previous calculated hidden states.

A defect of RNNs is that the early input features can be "forgotten" during the forwarding path which leads to an imbalanced understanding of the input features and

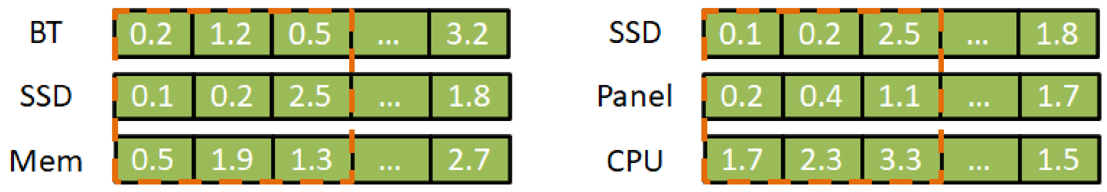


Figure 4.8: The same 3×3 filter (orange dot line) catches different system features in different input orders.

a reduction in prediction accuracy. However, in this project, we can utilize this defect to emphasize the important system features. This can be done by intentionally moving the important system features close to the output layer in the data pre-processing stage. Thus, the early input system features that have limited impact on the final results would be intentionally “forgotten” and the RNN model will tend to focus on catching the relationship among the late, important system features. In our numerous tests, we found that the original RNN would forget too much early information which leads to an unstable training process and relatively lower accuracy than MLP. Thus, we replace the ordinary RNN cell with a long-short term memory (LSTM) cell to overcome the issues. LSTM has a forget gate to selectively forget the aforementioned unimportant information. Therefore, part of the early input system features would be kept during the training process to increase the prediction accuracy.

4.8.3 Convolution Neural Network (CNN)

CNNs are a well-known machine learning model for capturing spatial locality. The structure of a CNN is flexible, but usually contains convolution layers, fully connected layers, pooling layers and residual layers. By applying convolution layers, the spatial locality of the input features are gradually integrated and the final target is predicted. A dataset that is suitable for learning by CNN usually has equally distributed spatial locality. For example, every pixel in an image is equally distributed and has the same spatial feature. In our dataset, that relationship is not satisfied. The size of the input matrix of a configuration is 15×12, as there are 15 system features and 1×12 word vector for each feature. The different order of system features would result in varying spatial locality, which can be demonstrated as follows. Suppose the convolution layer of a CNN model has a 3×3 filter. In the left figure of Figure 4.8, the first three system features are bluetooth (BT), SSD and memory (Mem), so the CNN model tends to integrate information from the word vectors of BT, SSD and Mem. If the first three system features are SSD, display panel and CPU as shown in the right part of Figure 4.8, the same CNN model catches completely different information and consequently, predicts different PnP values. Therefore, to utilize the CNN model, it is important to have the experienced prior information to provide reasonable input feature order.

4.9 Evaluation Results

4.9.1 Evaluation Methodology

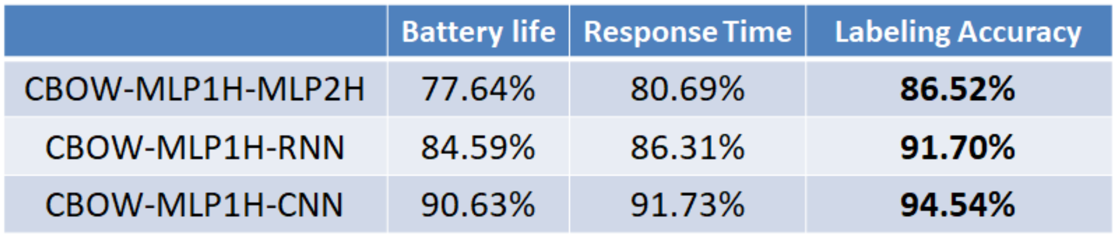
We tested many different structures and parameters for the proposed models, we will only present the one that provides the best accuracy for each of the models in the following sections. Our proposed MLP model, as aforementioned, has two hidden layers, with 128 hidden units and 64 hidden units, respectively. The proposed RNN model utilizes a LSTM cell and has 32 hidden units in its hidden layer. The CNN model has two convolution layers and 3 fully connected layers. Both convolution layers have 1×3 filter size, the first layer has 32 such filters and the second layer has 64 such filters. The number of hidden units in the three layers are 256, 128 and 64, respectively.

We introduce "accuracy" as a metric for evaluation purpose. The definition of accuracy in this project is as follows:

$$accuracy = \frac{1}{n}\sum_{n}(1 - \frac{|f_i - y_i|}{y_i})$$

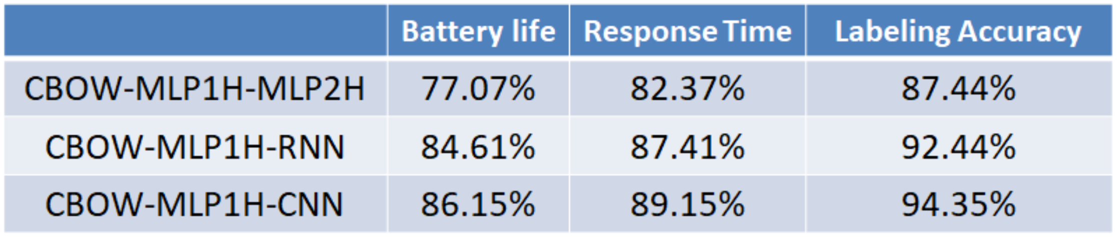
where n is the total number of data points, f_i and y_i are the predicted and measured PnP value for the i-th configuration. This accuracy reflects the averaged relative deviation of the predicted values from the measured values. To better understand the model efficiency, we also introduce "labeling accuracy" as a complement. Suppose the measured PnP value of a KEI in a system meets the pre-determined threshold of the KEI, then the data point is labeled as a pass, otherwise, the data point is labeled as a fail. If the predicted pass or fail label is the same as the measured pass or fail label for a given KEI, then this prediction is considered as a "hit" on that KEI. Then, we define the labeling accuracy as $\frac{Total\ hits}{Total\ tested\ KEIs\ on\ all\ the\ systems}$. In the following sections, we primarily use the two introduced metrics to evaluate our proposed models.

To evaluate our solution, we intentionally remove some component (memory and display panel as example) models and the configurations relating those models from the dataset. Then, we train the enhanced CBOW model first to get word vectors for all the hardware models of the system features. The word vectors of the removed hardware



	Battery life	Response Time	Labeling Accuracy
CBOW-MLP1H-MLP2H	77.64%	80.69%	**86.52%**
CBOW-MLP1H-RNN	84.59%	86.31%	**91.70%**
CBOW-MLP1H-CNN	90.63%	91.73%	**94.54%**

Figure 4.9: The accuracy and labeling accuracy for the proposed models with unseen memory models.



	Battery life	Response Time	Labeling Accuracy
CBOW-MLP1H-MLP2H	77.07%	82.37%	87.44%
CBOW-MLP1H-RNN	84.61%	87.41%	92.44%
CBOW-MLP1H-CNN	86.15%	89.15%	94.35%

Figure 4.10: The accuracy and labeling accuracy for the proposed models with unseen display panel models.

models are predicted by the hardware representing model (MLP with one hidden layer or MLP1H). Finally, all word vectors of a derivative system are fed into the PnP prediction model to get the PnP target.

4.9.2 Evaluation Results of Prediction Models

Figure 4.9 and 4.10 show the accuracy and labeling accuracy when there are unseen memory or display panel models. First of all, the two tables demonstrate that the three proposed PnP prediction models have relatively similar behavior with different unseen models. As mentioned, we moved 11 out of 52 memory models and 15 out of 75 panel models with their relating configurations to two testing datasets, then we train and test our NLP-like solution.

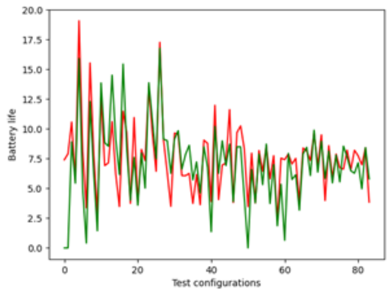


Figure 4.11: The measured (red) and predicted (green) curve for battery life in testing dataset with MLP2H.

It is clear that the MLP with two hidden layers (MLP2H) PnP prediction model performs worst among the three proposed models, with only around 87% labeling accuracy. This is mainly because the MLP model is too simple to catch the complicated data inter- and intra-relationship in our dataset. In our tests, we found that the number of hidden layers and the number of hiddens units in each layer have very limited impact to the prediction accuracy. That is, a more complicated MLP cannot catch the relationships as well. In Figure 4.11, we depict the measured and the predicted battery life values for part of the data points in testing dataset. The figure reveals that MLP2H only learns the coarse-grained trend of the testing dataset. In general, the MLP model only produces the baseline accuracy, but is also suitable for the scenario that no prior information is provided.

The RNN (LSTM) model achieves around 92% labeling accuracy in the two testing datasets which is notably better than MLP2H. According to our tests, the more hidden units in the RNN model, the higher accuracy that model is able to achieve. The labeling accuracy is improved from around 88% to 91% when the number of hidden units is increased from 4 to 32. However, more hidden units leads to training instability as

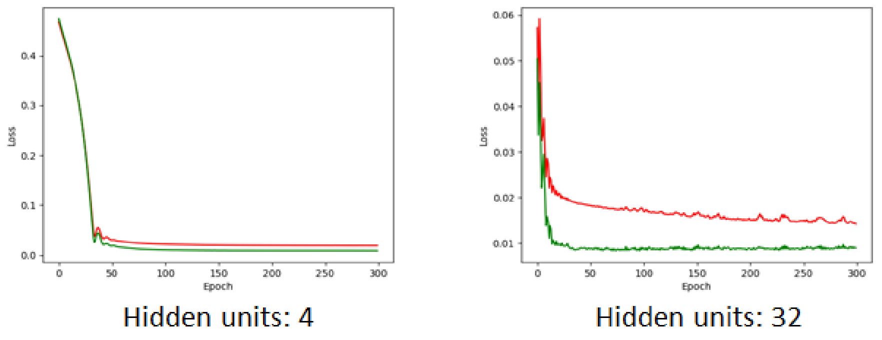


Figure 4.12: The loss-epoch curve for 4 and 32 hidden units in RNN.

depicted in Figure 4.12. The loss-epoch curve has more oscillation in the 32-unit RNN. Moreover, adding more hidden units has the potential for over-fitting issues in the future.

The proposed CNN model has the best labeling accuracy which is as high as around 94%. In our tests, we observe that the number and the size of convolution layers have a minor influence on the prediction results, as do the fully connected layers. The filter size, on the contrary, affects the predicted PnP results significantly. When we reduce the filter size from 7×3 to 1×3, the accuracy on the responsiveness dataset increases from 81.07% to 91.73%. This is because the height size impacts how many different pieces of component information are combined. For example, filter size 7×3 implies that every 7 neighboring component information would be integrated together and filter size 1×3 implies that no component information would be integrated. One important observation is that the influence of filter size on the responsiveness dataset and the battery life dataset is reversed. A larger filter size in the battery life dataset provides better accuracy. This observation implies that the information of hardware components should be considered comprehensively to predict battery life value while the information should be split when predicting response time.

Chapter 5: Conclusions

In the past a few years, machine learning algorithms are widely exploited in various fields including computer system design and optimization. Our EquiNox work suggests that the application of reinforcement learning models such as Monte Carlo Tree Search (MCTS) is capable of speeding up the search efficiency in huge design space without any prior experience. Our MCTS model produces the near-optimal result by assessing only 0.05% design space. The NLP-like solution introduces a novel point of view to resolve PnP prediction problem. The string data can be transformed to word vectors by applying an augmented CBOW model. The word vectors of the unseen component models are predicted by using the component specs with the proposed MLP1H model which also can be replaced to a more complicated model if necessary. The proposed PnP prediction model can be MLP2H, RNN or CNN where RNN and CNN provides exceptionally good prediction accuracy. Different ML models catch different information from the input features. In conclusion, machine learning algorithms are promising when applied to computer system optimization, which calls for more research on this field in the future.

Bibliography

[1] High-bandwidth memory (hbm) reinventing memory technology. `www.amd.com/Documents/High-Bandwidth-Memory-HBM.pdf`, Aug 2016.

[2] Samsung begins mass producing world's fastest dram based on newest high bandwidth memory (hbm) interface. `https://news.samsung.com/global/samsung-begins-mass-producing-worlds-fastest-dram-based-on-newest-high-bandwidth-memory-hbm-interface`, Jan 2016.

[3] Dennis Abts, Natalie D Enright Jerger, John Kim, Dan Gibson, and Mikko H Lipasti. Achieving predictable performance through better memory controller placement in many-core cmps. *ACM SIGARCH Computer Architecture News*, 37(3):451–461, 2009.

[4] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. *arXiv preprint arXiv:1902.01724*, 2019.

[5] Peter E Bailey, David K Lowenthal, Vignesh Ravi, Barry Rountree, Martin Schulz, and Bronis R De Supinski. Adaptive configuration selection for power-constrained heterogeneous systems. In *2014 43rd International Conference on Parallel Processing*, pages 371–380. IEEE, 2014.

[6] Ali Bakhoda, John Kim, and Tor M Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of the 43rd annual IEEE/ACM International Symposium on Microarchitecture (MICRO'10)*, pages 421–432, 2010.

[7] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09)*, pages 163–174, 2009.

[8] Bahareh Banijamali, Chien-Chia Chiu, Cheng-Chieh Hsieh, Tsung-Shu Lin, Clark Hu, Shang-Yun Hou, Suresh Ramalingam, Shin-Puu Jeng, Liam Madden, and Doug CH Yu. Reliability evaluation of a cowos-enabled 3d ic package. In *63rd IEEE Electronic Components and Technology Conference (ECTC'13)*, pages 35–40, 2013.

[9] Justin A Boyan and Michael L Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*, pages 671–678. Citeseer, 1994.

[10] Peter Braun and Heiner Litz. Understanding memory access patterns for prefetching. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*, 2019.

[11] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1), 2012.

[12] Raghunandan Chaware, Kumar Nagarajan, and Suresh Ramalingam. Assembly and reliability challenges in 3d integration of 28nm fpga die on a large high density 65nm passive interposer. In *62nd IEEE Electronic Components and Technology Conference*, pages 279–283, 2012.

[13] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization (IISWC'09)*, pages 44–54, 2009.

[14] W. J. Dally. Express cubes: improving the performance of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, 1991.

[15] Joeri De Vos, L Bogaerts, T Buisson, C Gerets, G Jamieson, K Vandersmissen, A La Manna, and E Beyne. Key elements for sub-50μm pitch micro bump processes. In *63rd IEEE Electronic Components and Technology Conference*, pages 1122–1126, 2013.

[16] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Express cube topologies for on-chip interconnects. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'09)*, pages 163–174, 2009.

[17] SY Hou, W Chris Chen, Clark Hu, Christine Chiu, KC Ting, TS Lin, WH Wei, WC Chiou, Vic JC Lin, Victor CY Chang, et al. Wafer-level integration of an advanced logic-memory system through the second-generation cowos technology. *IEEE Transactions on Electron Devices*, 64(10):4071–4077, 2017.

[18] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W Keckler. Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems. *ACM SIGARCH Computer Architecture News*, 44(3):204–216, 2016.

[19] Hyunjun Jang, Jinchun Kim, Paul Gratz, Ki Hwan Yum, and Eun Jung Kim. Bandwidth-efficient on-chip interconnect designs for gpgpus. In *Proceedings of the 52nd Annual Design Automation Conference*, page 9. ACM, 2015.

[20] JEDEC. High bandwidth memory (hbm) dram. *JESD235*, 2013.

[21] Natalie Enright Jerger, Ajaykumar Kannan, Zimo Li, and Gabriel H Loh. Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free? In *Proceedings of the 47th annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*, pages 458–470, 2014.

[22] Nan Jiang, James Balfour, Daniel U Becker, Brian Towles, William J Dally, George Michelogiannakis, and John Kim. A detailed and flexible cycle-accurate network-on-chip simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*, pages 86–96, 2013.

[23] Daniel A Jiménez. An optimized scaled neural branch predictor. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 113–118. IEEE, 2011.

[24] Daniel A Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206. IEEE, 2001.

[25] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H Loh. Enabling interposer-based disintegration of multi-core processors. In *Proceedings of the 48th annual IEEE/ACM International Symposium on Microarchitecture (MICRO'15)*, pages 546–558, 2015.

[26] Muhammad Khan and Min Sung Kim. Damascene process and chemical mechanical planarization, 2015.

[27] Hanjoon Kim, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. Providing cost-effective on-chip network bandwidth in gpgpus. In *30th IEEE International Conference on Computer Design (ICCD'12)*, pages 407–412, 2012.

[28] J. Kim, J. Balfour, and W. Dally. Flattened butterfly topology for on-chip networks. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 172–182, 2007.

[29] Kyung Hoon Kim, Rahul Boyapati, Jiayi Huang, Yuho Jin, Ki Hwan Yum, and Eun Jung Kim. Packet coalescing exploiting data redundancy in gpgpu architectures. In *Proceedings of the International Conference on Supercomputing (ICS'17)*, page 6. ACM, 2017.

[30] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *Computer Architecture Letters*, 15(1):45–49, 2016.

[31] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[32] John H Lau. The future of interposer for semiconductor ic packaging. *Chip Scale Rev*, 18(1):32–36, 2014.

[33] Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Kang Seol Lee, Sang Jin Byeon, Jae Hwan Kim, Jin Hee Cho, Jaejin Lee, and Jun Hyun Chun. A 1.2 v 8 gb 8-channel 128 gb/s high-bandwidth memory (hbm) stacked dram with effective i/o test circuits. *IEEE Journal of Solid-State Circuits*, 50(1):191–203, 2015.

[34] HY Li, HM Chua, FX Che, Alastair David Trigg, KH Teo, and S Gao. Redistribution layer (rdl) process development and improvement for 3d interposer. In *13th IEEE Electronics Packaging Technology Conference (EPTC'11)*, pages 341–344, 2011.

[35] Li Li, Peng Su, Jie Xue, Mark Brillhart, John Lau, PJ Tzeng, CK Lee, CJ Zhan, MJ Dai, HC Chien, et al. Addressing bandwidth challenges in next generation high performance network systems with 3d ic integration. In *62nd IEEE Electronic Components and Technology Conference (ECTC'12)*, pages 1040–1046, 2012.

[36] Daniel Lo, Taejoon Song, and G Edward Suh. Prediction-guided performance-energy trade-off for interactive applications. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 508–520, 2015.

[37] Mie Matsuo, Nobuo Hayasaka, Katsuya Okumura, Eiichi Hosomi, and Chiaki Takubo. Silicon interposer technology for high-density package. In *50th IEEE Electronic Components and Technology Conference (ECTC'00)*, pages 1455–1459, 2000.

[38] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*, 53(2):184–198, 2018.

[39] NVIDIA. Nvidia tesla p100: The most advanced data center accelerator. `https://www.nvidia.com/en-us/data-center/tesla-p100/`.

[40] NVIDIA. Cuda code samples. `https://developer.nvidia.com/cuda-code-samples`, 2018.

[41] Ennis T Ogawa, Ki-Don Lee, Volker A Blaschke, and Paul S Ho. Electromigration reliability issues in dual-damascene cu interconnections. *IEEE Transactions on reliability*, 51(4):403–419, 2002.

[42] Gung-Yu Pan, Jing-Yang Jou, and Bo-Cheng Lai. Scalable power management using multilevel reinforcement learning for multiprocessors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 19(4):1–23, 2014.

[43] Python. The python tutorial. `https://docs.python.org/release/3.7.3/tutorial/index.html`.

[44] Venkata Yaswanth Raparti and Sudeep Pasricha. Memory-aware circuit overlay nocs for latency optimized gpgpu architectures. In *17th IEEE International Symposium on Quality Electronic Design (ISQED'16)*, pages 63–68, 2016.

[45] Md Farhadur Reza, Tung Thanh Le, Bappaditya De, Magdy Bayoumi, and Dan Zhao. Neuro-noc: Energy optimization in heterogeneous many-core noc using neural networks in dark silicon era. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[46] Kirk Saban. Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency. *Xilinx, White Paper*, 1:1–10, 2011.

[47] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham,

Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[48] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[49] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12)*, pages 201–210, 2012.

[50] Stephen J Tarsa, Chit-Kwan Lin, Gokce Keskin, Gautham Chinya, and Hong Wang. Improving branch prediction by modeling global history with convolutional neural networks. *arXiv preprint arXiv:1906.09889*, 2019.

[51] Elvira Teran, Zhe Wang, and Daniel A Jiménez. Perceptron learning for reuse prediction. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[52] Chien-Fu Tseng, Chung-Shi Liu, Chi-Hsi Wu, and Douglas Yu. Info (wafer level integrated fan-out) technology. In *66th IEEE Electronic Components and Technology Conference (ECTC'16)*, pages 1–6, 2016.

[53] Hua Wang, Xinbo Yi, Ping Huang, Bin Cheng, and Ke Zhou. Efficient ssd caching by avoiding unnecessary writes using machine learning. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[54] Brett Williams, Deborah Florence, Hormazdyar Dalal, Krishna Gunturu, Mark Nelson, and Chuck Belisle. Rdl manufacturing for flip chip packaging. In *IEEE Workshop on Microelectronics and Electron Devices (WMED'05)*, pages 28–31, 2005.

[55] Jae-Yeon Won, Xi Chen, Paul Gratz, Jiang Hu, and Vassos 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)*, pages 308–319. IEEE, 2014.

[56] Xilinx. Cost and transceiver optimized fpgas. `https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html`.

[57] Jieming Yin, Zhifeng Lin, Onur Kayiran, Matthew Poremba, Muhammad Shoaib Bin Altaf, Natalie Enright Jerger, and Gabriel Loh. Modular routing design for chiplet-based systems. In *International Symposium on Computer Architecture*, 2018.

[58] Yuan Zeng and Xiaochen Guo. Long short term memory based hardware prefetcher: a case study. In *Proceedings of the International Symposium on Memory Systems*, pages 305–311, 2017.

[59] Xia Zhao, Sheng Ma, Chen Li, Lieven Eeckhout, and Zhiying Wang. A heterogeneous low-cost and low-latency ring-chain network for gpgpus. In *34th IEEE International Conference on Computer Design (ICCD'16)*, pages 472–479, 2016.

[60] Xia Zhao, Sheng Ma, Yuxi Liu, Lieven Eeckhout, and Zhiying Wang. A low-cost conflict-free noc for gpgpus. In *Proceedings of the 53rd Annual Design Automation Conference*, page 34. ACM, 2016.

[61] Amir Kavyan Ziabari, José L Abellán, Yenai Ma, Ajay Joshi, and David Kaeli. Asymmetric noc architectures for gpu systems. In *9th IEEE/ACM International Symposium on Networks on Chip (NoCS'15)*, page 25, 2015.