Taming Extreme Heterogeneity via Machine Learning based Design of Autonomous Manycore Systems

Special Session Paper

Paul Bogdan[1], Fan Chen[3], Aryan Deshwal[2], Janardhan Rao Doppa[2], Biresh Kumar Joardar[2], Hai (Helen) Li[3], Shahin Nazarian[1], Linghao Song[3], and Yao Xiao[1]
University of Southern California[1], Washington State University[2], and Duke University[3]
{pbogdan,shahin,xiaoyao}@usc.edu,{jana.doppa,biresh.joardar,aryan.deshwal}@wsu.edu,{hai.li,fan.chen,ls334}@duke.edu

ABSTRACT

To avoid rewriting software code for new computer architectures and to take advantage of the extreme heterogeneous processing, communication and storage technologies, there is an urgent need for determining the right amount and type of specialization while making a heterogeneous system as programmable and flexible as possible. To enable both programmability and flexibility in the heterogeneous computing era, we propose a novel complex network inspired model of computation and efficient optimization algorithms for determining the optimal degree of parallelization from old software code. This mathematical framework allows us to determine the required number and type of processing elements, the amount and type of deep memory hierarchy, and the degree of reconfiguration for the communication infrastructure, thus opening new avenues to performance and energy efficiency. Our framework enables heterogeneous manycore systems to autonomously adapt from traditional switching techniques to network coding strategies in order to sustain on-chip communication in the order of terabytes. While this new programming model enables the design of self-programmable autonomous heterogeneous manycore systems, a number of open challenges will be discussed.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Model of computation, self-programming computing architectures, manycore systems, machine learning, ReRAM, processing-in-memory, autonomous design optimization

ACM Reference Format:
Paul Bogdan[1], Fan Chen[3], Aryan Deshwal[2], Janardhan Rao Doppa[2], Biresh Kumar Joardar[2], Hai (Helen) Li[3], Shahin Nazarian[1], Linghao Song[3], and Yao Xiao[1]. 2019. Taming Extreme Heterogeneity via Machine Learning based Design of Autonomous Manycore Systems: Special Session Paper. In *International Conference on Hardware/Software Codesign and System Synthesis Companion (CODES/ISSS '19 Companion), October 13–18, 2019, New York, NY, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3349567.3357376

1 INTRODUCTION

Edge, fog, and exascale computing (EC) are essential for validating scientific theories and developing revolutionary biological, nano- or neuro-technologies to tackle 21st century challenges (e.g., precision medicine, energy crisis, climate change, and smart, safe and secure cities). Advanced scientific and engineering investigations call for revolutionary EC design approaches that break the hardware-software boundary and propose breakthroughs for heterogeneous scalable computing platforms and memory storage. For example, graph analytics decodes the links among heterogeneous data streams and offers insights to ensure accurate prediction and decision-making. Inferring causal and higher-order complex relationships from unstructured time-varying data calls for a paradigm shift in EC design.

Today's general-purpose manycore computing systems are widely used in industry and research. However, due to synchronization overhead, load imbalance, and resource sharing, parallel programming models such as pthreads and OpenMP can exacerbate application performance, making it non-ideal to design scalable systems. Moreover, these platforms are based primarily on the stored-program computer concept and are implemented using von Neumann computing architecture, which separates the computation and storage into two distinct components connected by an off-chip bus. Data is frequently exchanged between the computing units (*e.g.*, CPUs/GPUs) and the memory units. Due to the disparity between processing and memory technologies, the latency and energy of data movement are generally a few orders of magnitude higher than the computation in both servers [29] and mobile devices [43]. Hence, data communication cost dominates computation cost and becomes the major bottleneck for improving the overall system performance and execution efficiency, which is a phenomenon widely known as the "*memory wall*". The situation is more severe when dealing with big-data applications (*e.g.*, training of deep neural networks) with intense demand in computation and memory accesses [8, 9, 41].

We therefore propose several state-of-the-art design methodologies and architectures to address the afore-mentioned issues. Section 2 introduces some challenges for edge, fog, and exascale computing. Section 3 describes mathematical and algorithmic tools for building dynamic MoCs that support complex reasoning about the nature and type of computations, their interactions with the data (numerical algorithms) and their memory requirements, thus enabling new pathways for performance and energy optimization. Section 4 presents memory-centric architectures to address the memory wall issue. Section 5 introduces machine learning based design space exploration and optimization algorithms for optimizing heterogeneous manycore systems. Section 6 presents dynamic resource management using machine learning techniques. Section 7 concludes the paper and offers some future directions.

2 EDGE, FOG, AND EXASCALE COMPUTING CHALLENGES

EC desiderata can be achieved via a cross-layer understanding of performance and energy inefficiency from high-level algorithms (including programming languages, compilers, and software libraries) to lower-level hardware and operating system. Towards this end, the EC challenges coming from the application-algorithm side are: (1) How to decide and design the EC (deep machine) hierarchy in terms of hardware partitioning (how many cores per tile, how to interconnect tiles, blocks, units, sockets, modules, racks) and heterogeneous memory allocation (e.g., DRAM, SRAM, MRAM)? How to determine the data layouts and execution schedules on this hierarchical EC architecture for running applications/programs? (2) How to couple the concurrency structure and dynamic behavior of applications with opportunities for dynamic voltage and frequency scaling (DVFS) on die? Can we develop new models of computation (MoC) that enable the computing architectures and applications to self-program and self-manage to achieve high performance and energy efficiency? (3) How can we construct a MoC for emerging applications (from scarce or possibly poorly written software code) to better understand the concurrency structure (e.g., data and inter-task dependencies) and dynamics (e.g., data reuse, data movement, fine-grained synchronization)? Can this MoC enable the identification of true computation and communication requirements (management of memory and data movement)? Can the MoC help in minimizing the data movement and I/O operations?

3 MOC AND ALGORITHMS FOR AIECS

Imagine a future in which scientists formulate algorithms to process unseen (and potentially) uncertain type of streaming data yet they leave their realization to an autonomous intelligent exascale computing system (AIECS) [62]. Accomplishing this grand vision requires a radical paradigm shift in designing AIECS capable of exploiting extreme heterogeneity through artificial intelligence and machine learning (AI/ML) strategies for addressing the following grand challenges:

(1) Self-programming: How can an automated intelligence integrated within future computing systems discover the MoCs that enable the determination of optimal degree of parallelism and support heterogeneous processing at run-time from existing code written for past or current programming models or architectures? How can we construct a hierarchical MoC representation of massive parallelism in evolvable applications and codes [56, 59]?

(2) Self-introspection [55]: How to endow computing platforms with performance analysis capabilities that inform online self-optimizing intelligence of potential bottlenecks for taking proactive measures [55]?

(3) Self-reconfiguration and self-optimization [19, 24, 45, 57, 58]: What MoCs, metrics and algorithmic tools are required to determine which code regions perform well on which processor type and adaptively map codes to hardware resources while also supporting the reconfiguration of hardware to match the input problem and data dependent variations? How to endow AIECS with self-learning and self-optimizing capabilities to capture the nuances of complex memory hierarchies, support efficient data and instruction movement and support code changes in emerging evolvable applications?

Designing efficient programmable heterogeneous computing architectures partially relies on understanding the sophisticated high-level languages. The structure of each application is characterized by its size, composition and task topology [59]. Usually, static compiler analysis such as data and control dependency analysis is required to construct the corresponding task graph. However, such static compilation fails to capture the dynamic nature of memory operations and loop count, making it difficult to balance the workloads efficiently. This requires a compiler to build a dynamically learned MoC during the execution time. Consequently, the goal of this programming model is to build an application abstraction that enables the optimal parallelization of applications running on heterogeneous computing architectures. This captures run-time behaviors of tasks in applications and on-chip communication, which could alleviate three primary issues in performance degradation: load imbalance, resource sharing, and synchronization overhead [56]. Each task i can be represented as a graph of dynamic instruction traces which are classified as i. The structure of tasks can be denoted as inter-task and intra-task communication, i.e., data dependency and control dependency. Therefore, we introduce the following definitions to mathematically characterize the programming model.

Definition 1: A dynamic task $P(i)$ is defined as $P(N(t), E(t), I, O, T|\text{type}=i)$ where T represents the time horizon t starting from 0; $N(t)$ represents a sequence of nodes (instructions) generated for this task over time t; $E(t)$ represents a collection of edges (dependencies among instructions) inside each task; I and O represent the finite disjoint sets of inputs and outputs for each task, respectively.

I and O are finite alphabet of inputs and outputs to provide a task IO abstraction for inter-task dependency modeling. We use English alphabet from a to z to prevent any architecture-specific assumptions such as the number and size of registers. This allows us to run applications with this model without concerning the specific machine architecture. Time horizon T is introduced to capture the true dependencies of instruction traces including memory operations. Due to its dynamic compilation, $N(t)$ and $E(t)$ are required to represent the various number of nodes and edges over time T. We associate a task with a specific type, indicating all of instructions $N(t)$ are inside the task, which helps to mathematically define the task formulation problem.

For instance, we can consider $N(t)$ as a sequence of instructions arranged in an interconnected two-layer network [54]. One layer contains compute instructions to represent **model of computation**. This could facilitate designers to implement more efficient heterogeneous processing elements. The other layer consists of memory operations, i.e., load-store instructions, which forms **model**

of communication to explore the design space of heterogeneous memory systems and design the memory architecture.

Definition 2: An application trace *Tr* is defined as a sequence of instructions (computation and communication) running with different representative inputs.

Definition 3: An application graph *AG* can be mathematically described as *AG(N'(P), E'(P), C)* where *N'(P)* represents the number of tasks; *E'(P)* represents the number of inter-task edges; and most importantly, *C* represents an autonomous intelligent classifier to identify communities of strongly interacting instructions (tasks).

The autonomous intelligence sitting between software and hardware can be regarded as a function to map instructions into different types in definition 1. Mathematically speaking,

$$C : N(t) \longrightarrow type = i, \forall i \qquad (1)$$

This intelligence determines (1) which instructions could be combined into a processing community or task under different objectives (e.g., performance maximization or energy minimization); (2) the number of tasks generated due to synchronization overhead. The novelty is that we associate the mathematical model of each application with an intelligence to generate suitable tasks in terms of different systems, which can be implemented as community detection in complex network or machine learning techniques for traffic-aware NoC based platforms or self-programmable heterogeneous computing systems, respectively.

Therefore, the programming model decides the choice of the intelligence *C*. The goal is to choose the best intelligence model *C* to maximize performance improvement and on-chip traffic reduction in programmable heterogeneous computing architectures. Mathematically, we can formulate the problem (task extraction from probabilistic reasoning of compiler analysis and task graph partitioning) as the following:

Given an application trace *Tr*, **find** the intelligence *C* to **maximize** a user-defined function *f*

$$f = w_1 PS(AG) + w_2 ER(AG) + w_3 TC(AG) + w_4 LB(AG) \quad (2)$$

where *PS, ER, TC, LB* represent performance speedup, energy reduction, traffic congestion, and load balancing, respectively; w_1, w_2, w_3, and w_4 are user-defined parameters.

The following sections relate it with the traditional programming model and describe a number of intelligence models we can consider.

3.1 Threads and Processes

In traditional parallel programming [12, 35, 46], pthreads in POSIX and OpenMP are widely utilized to develop performance efficient applications for platforms from manycore systems to supercomputers. Our programming model is trying to assign different types to different threads or processes which lead to parallel execution. For example, in OpenMP, the pragma 'omp parallel for' is used to split up loop iterations among threads. The programming model captures it by assigning different types to different loop traces. However, there are two main issues: First, locks are commonly deployed to prevent different threads from updating the shared variables simultaneously. If threads are not properly spawned, this could potentially exacerbate performance due to synchronization overhead. Second, it is not aware of how much information is transferred among threads. Sometimes, this could cause traffic congestion or deadlock issue. Therefore, a more clever approach to designing the classifier in parallel programming is required to improve our objectives.

3.2 Optimal Parallelization Discovery

Applications can be described as weighted directed acyclic graphs. Nodes represent computations/memory load-store instructions, and edges represent dependencies. Graph partitioning remains vital in parallel computing to divide the computations among cores. One approach guarantees the number of clusters produced such as k-way partitioning [4]. However, in complex networks, community detection [21] is widespread in identifying communities with dense connection internally and sparser connections between groups. Inspired by complex network theory, we can find optimal parallelism from community structures by an optimization framework. In other words, one realization of traffic reduction on chips is by minimizing the amount of messages transferred among different communities/tasks.

Energy consumption on the interconnect has reached up to 40% of the total energy. One method to reduce the amount of energy is to transfer fewer messages among cores. On the other hand, there is a diminishing return in performance improvement over the increasing number of cores due to synchronization overhead and load imbalance. Therefore, in order to overcome these issues, an optimization framework [56] is proposed to automatically parallelize complex programs, while balancing the workloads among cores and constraining the cluster count approximately equal to the core count at the same time. Therefore, identification of optimal parallel community structures can be achieved by maximizing the following function.

$$Q = \sum_{c=1}^{n} [\frac{W^{(c)}}{W} - (\frac{S^{(c)}}{2W})^2] - \lambda_1 R_1 - \lambda_1 R_2 \qquad (3)$$

$$R_1 = \frac{1}{W^2} \sum_{c=1}^{n} [W_c - W_{(c+1) \bmod n}]^2 \qquad (4)$$

$$R_2 = \frac{1}{n^2} (n - N)^2 H(n - N) \qquad (5)$$

where n represents cluster count; N represents core count; $W^{(c)}$ denotes the sum of weights connected within a cluster c ($W^{(c)} = \sum_{i \in c} \sum_{j \in c} w_{ij}$); W is the sum of all weights in a graph ($W = \sum_i \sum_j w_{ij}$); $S^{(c)}$ is the sum of all weights adjacent to a cluster c ($W^{(c)} = \sum_{i \in c} \sum_{j \notin c} w_{ij}$); λ_1 and λ_2 are regularization parameters; $H(x)$ is Heaviside step function ($H(x) = \int_{-\infty}^{x} \delta(s) ds$); $\delta(x)$ is Dirac delta function.

The function in eq(3) can be regarded as a specific intelligent model C which maps instructions to different clusters. The first term discovers the parallel dense clusters with sparse interdependent connection. The second term is a regularization term to make sure the workloads between pairs of clusters (W_c and $W_{(c+1) \bmod n}$) are balanced. The third term is another term to confine cluster size n to core count N.

4 MEMORY-CENTRIC ARCHITECTURES

A key promising approach of solving the memory wall issue is to shift the previous processor-centric paradigm towards memory-centric, which enables the opportunity of performing processing close to or in memory and avoid unnecessary data movement. As demonstrated in Figure 1(b), the memory can be developed as an accelerator such that data is processed in-situ where it is stored. Processing-in-memory (PIM) accelerator eliminates a large amount of data movement, and thereby, significantly reduces the processing latency and system energy. Various PIM architectures have recently been adopted to multiple different applications [1, 5–7, 10, 20, 28, 49, 51–53].

In this section, we present our recent research that aims to practically enable computation in memory for deep learning (DL) applications. We first review the in-memory processing unit designs, followed by our approach of a parallel dataflow for scalable accelerators. Specifically, we proposed a layer-wise pipeline for PIM architectures incorporating the inter-layer execution and intra-layer parallelism to accelerate the processing of DNNs. Then we consider DNNs and large-scale dataset that cannot fit into a single accelerator, which is the inevitable consequence of the ever growing complexity at the algorithm level. We present a novel communication model to investigate the fundamental mechanism of data movement in such cases. Based on the investigation, we present a hybrid parallelism scheme, which partitions the feature map and the weight across layers to achieve high performance and high energy efficiency on an array of PIM accelerators.

4.1 In-Memory-Processing Engines

As illustrated in Figure 1(c), the processing of DNNs are typically executed recursively on the same computational blocks (*a.k.a.*, processing engines, or PEs) which are designed to support the basic vector-matrix multiplication (VMM) operations. PIM accelerators use logic inside memory systems or the memory itself to implement PEs. The former approach is primarily used in traditional CMOS-based designs, while the implementations based on emerging technologies typically adopt the latter by taking advantage of the computation capabilities of new types of devices. Here, we provide a brief review of PE designs for VMM based on the resistive random access memory (ReRAM), and refer interested readers to other emerging technologies based designs [33, 47, 64].

Hu *et al.* [22] proposed to map the matrix weights to the conductance states in a ReRAM array and encode the voltages on the wordlines as the input vector. Then the accumulated currents from the bitlines are the results of the VMM. That is the primary paradigm for the computation of VMM in ReRAM, *i.e.* ReRAM VMM processing engines. In 2016, Hu *et al.* [23] fabricated a 4×4 ReRAM VMM PE. Yan *et al.* [63] implemented VMM PE by integrating 64Kb binary ReRAM and scalable nonlinear spike-based data converter monolithically, which fuses analog/digital conversion and activation function by leveraging its nonlinear working region. Based on the ReRAM VMM PEs, accelerators for the multilayer perceptron (MLP), *a.k.a.* fully connected (FC) layer, can be designed. RENO [39], building on the ReRAM VMM PEs, is an on-chip accelerators for fully connected neural networks. An instruction set architecture (ISA) was designed to coordinate the execution of the CPU and the accelerator in [39]. However, the performance, energy efficiency and supported applications of on-chip design are very limited.

4.2 Optimal Data Placement

In PIM, memory bandwidth inside each vault is much higher than that between different vaults. Due to this unconventional architecture compared to DRAM systems, three issues need to be addressed before this technology is adopted in industries: (1) How to design an MoC to exploit the new PIM architecture? (2) How to scale up future PIM systems to have hundreds of vaults? (3) How to place data on different vaults to reduce data movement and utilize internal memory bandwidth?

Prometheus [54] describes an optimization framework to decide optimal data placement in the context of PIM systems. First, it models both computation and communication for each application to exploit the PIM architecture. It constructs a two-layer network where one layer represents model of computation with compute nodes and the other layer represents model of communication with memory load-store instructions. In this way, we can design a specialized logic accelerator for each vault and decide the optimal data placement. Second, it presents a scalable NoC based PIM system to efficiently transfer packets to the destination vaults. Third, an optimization model is proposed to identify community structures from the network as follows. Each community structure consists of one layer of model of computation and one layer of model of communication to guide us for logic design and data placement.

$$max \quad F = Q - \alpha LB \tag{6}$$

$$Q = \frac{1}{2W} \sum_{ij} [W_{ij} - \frac{s_i s_j}{2W}] \delta(C_i, C_j) \tag{7}$$

$$LB = \frac{1}{2W} \sum_{\substack{1 \le u \le n_c \\ 1 \le v \le n_c \\ u \ne v}} |W_u - W_v| \delta(d_u, d_v) \tag{8}$$

where W is the sum of the total weights; W_i is the sum of weights in the community i; W_{ij} is the weight between nodes i and j; s_i, the strength of a node i, is the sum of the weights of edges adjacent to the node i; C_i is the community to which node i belongs; n_c is the number of communities; d_i is the depth of community i; $\delta(i, j)$ equals 1 when $i = j$; α is the user-defined parameter to control load balancing.

One can extend this approach to consider the heterogeneous memory systems consisting of traditional DRAM, PIM, and NVM such as ReRAM and STT-RAM. Each emerging memory technology has its own benefits. For example, PIM has better memory bandwidth but also higher power. NVM is much denser compared to others. From our model of computation and communication, Deciding the types and sizes of memory technologies for each cluster under the objective of maximizing performance is an open question waiting to be resolved.

4.3 Parallel Dataflow for Scalable Accelerators

Neural networks are inherently parallel algorithms with the potential for data sharing. Our goal is to organize PEs and on-chip memory in a parallel or pipelines fashion in order to exploit the parallelism in DNNs. According to our research, there are *inter-layer* and *intra-layer*, two levels of parallelism in DNN applications.

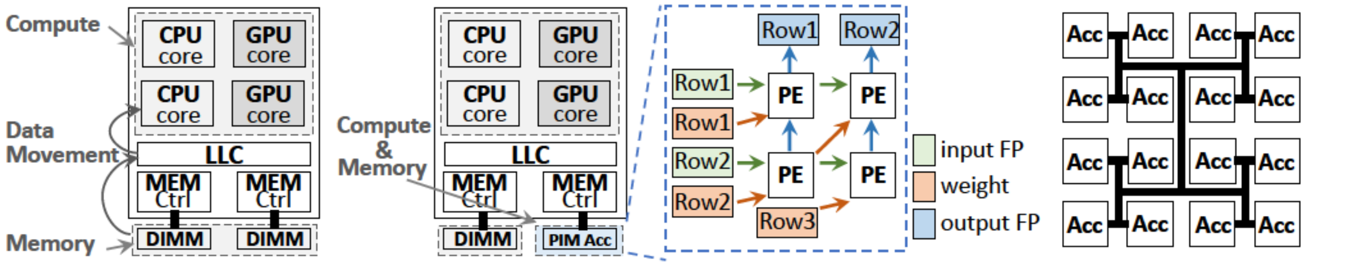


Figure 1: (a) Von Neumann architecture and memory wall. Data movement is 2∼3 orders of magnitude the latency/energy consumption of computation; (b) PIM accelerator; (c) PE array and dataflow; (d) acclerator array in an H-tree.

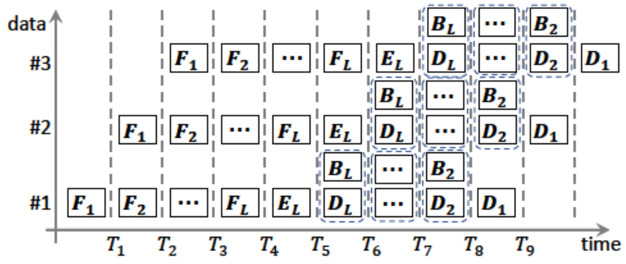


Figure 2: Inter-layer pipeline.

Inter-layer Parallelism. Previous work [9, 49] pipeline the processing of DNNs to improve system throughput. However, they focus only on the inference phase while not being able to support more sophisticated and challenging training phase which involves weight updates and complex data dependencies. We propose a DNN model parallel scheme in which multiple training data can be processed in an efficient pipeline. The proposal is based on an important observation: input data are normally processed in a large batch size B (*e.g.*, 128). The weights applied on the inputs within a batch remain unchanged which will be updated only at the end of a batch. Therefore, no dependency exists among data inputs in each batch. Based on this observation, we demonstrate the inter-layer pipeline execution for a 3-layer DNN model in Figure 2. In this example, the calculation of the forward path processing, backward error propagation and weight partial derivatives for layer i are respectively denoted as F_i, B_i, D_i $(i \in 1, 2, 3)$. In the execution, a new input can enter the pipeline every cycle within a batch. At the end of a batch, a new input belonging to the next batch cannot enter the pipeline immediately, but wait until all inputs in the current batch are processed and the weights are updated. In another word, a new batch has to wait the "tai" of the previous batch to drain from the pipeline.

G: Parallelism Granularity; L: Number of Layers; B: Batch Size; N: Total Number of Input Images.		
	Non-pipelined	Pipelined
Forward Cycles	LN	(N/B)(2L+B+1)
Backward Cycles	$(L+1)N+N/B$	
PE Clusters	$GL+G(2L-1)$	$GL+G(L-1)+BL$

Table 1: Latency and cost of the proposed architecture.

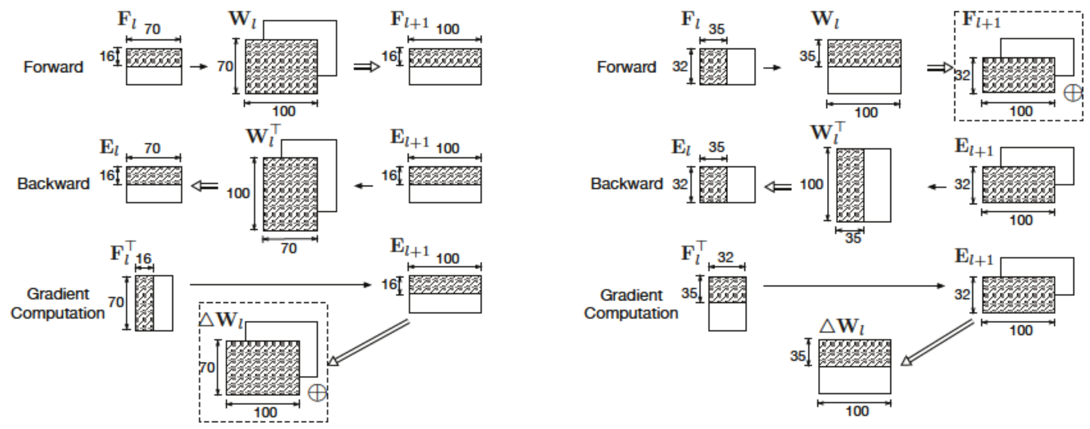
Intra-layer Parallelism. Because of the imbalance in compute/memory requirements for each layer, we discussed how to improve system performance by adjusting the degree of parallelism for each layer. We define an auxiliary metric called *parallelism granularity*, denoted as G, indicating the number of the duplicated copies of PE clusters that store the same weights. Essentially, *parallelism granularity* allows to explore the trade-off between the hardware resource and performance. A good trade-off involves a carefully chosen G. Our experimental results show that the performance (*i.e.*, latency) increases monotonically with G, while the chip area increases. Therefore, choosing the suitable G to explore the balance between speedup and area is critical. Table 1 compares the latency and hardware overhead of non-pipelined and the proposed pipelined architectures. More detailed discussion and explorations can be found in [52].

4.4 Hybrid Parallelism for Accelerator Arrays

As the complexity of deep learning models continues to increase, it is difficult to meet the throughput and energy requirements of the training procedure with a single accelerator. Figure 1(d) demonstrates an example of 16 accelerators connected as an H-tree. Previous works only considered the layer-wise parallelism within an accelerator in fine grain. For the first time, we conducted systematic studies to seek a coarse-grain parallelism among an array of accelerators.

Parallelism Types. Existing works can be divided into two categories based on parallel types: *data parallelism* (dp) [38] and *model parallelism* (mp) [16]. In *data parallelism*, data is partitioned into portions and run simultaneously on multiple accelerators with the same model copy; in *model parallelism*, the model is divided and distributed among multiple accelerators and each processes the same training data on the sub-model it holds.

Communication Model. Unlike the previous empirical approach [34], we developed a communication model that analytically explains the source and amount of communications. We consider a fully-connected layer with 70 input and 100 output neurons, respectively. Assume that there are two accelerators and the batch size is $B = 32$, Figure 3 illustrates the shapes of tensors held by the two accelerators. Each layer performs three multiplications for the forward, error backward, and gradient computation. In each multiplication, three tensors are involved. Thus, nine tensors in total need to be considered. Following the aforementioned intra-layer and inter-layer idea [52], we decouple the communication into two parts: *1) intra-layer communication* by kernel updates within a layer (marked by a ⊕ in Figure 3); and *2) inter-layer communication* by conversions of L and R tensors of feature maps and errors between



(a) data parallelism (b) model parallelism

Figure 3: Forward, Backward and Gradient Computation in (a) data parallelism and (b) model parallelism [51] *(Note that all of the rectangles with shadow lines are held by one accelerator and all of the white rectangles are held by the other.)*

layers. Due to the page limit, we omit the detailed analysis, but summarize the conclusion in Table 2. It shows that, for the DNN inference, the *data parallelism* is better because the *intra-layer communication* in inference is zero and the *inter-layer communication* of dp-dp is also zero. However, the conclusion does not hold for the DNN training, where the parallelism becomes a critical concern. A Naïve approach would be to get a parallelism type for each layer and enumerate all possibilities to determine the best performance, resulting in $O(2^N)$ time complexity.

	Type	Communication Amount
Intra-Layer	dp	$\mathbb{A}(\triangle W_l)$
	mp	$\mathbb{A}(F_{l+1})$
Inter-Layer	dp-dp	0
	dp-mp	$0.25\mathbb{A}(F_{l+1}) + 0.25\mathbb{A}(E_{l+1})$
	mp-mp	$0.5\mathbb{A}(E_{l+1})$
	mp-dp	$0.5\mathbb{A}(E_{l+1})$

Table 2: Communication cost in data parallelism (dp) and model parallelism (mp).

Hybrid Parallelism. We propose a more practical partitioning algorithm which reduces the $O(2^N)$ time complexity to $O(N)$. The proposed approach is based on three observations: *1)* the parallel mode of each layer may only be *data parallelism* or *model parallelism*; *2)* the inter-layer traffic depends only on two adjacent layers; and *3)* the *intra-layer communication* is exclusively dependent on the parallelism of that layer, completely independent of other layers. Thus, we can use a layer-wise dynamic programming method to search for the optimal partition of each layer. Specifically, *intra-layer communication* of dp and mp and *inter-layer communication* of dp-dp, mp-dp, dp-mp, mp-mp are looked up in Table 2. Based on these data, the minimum accumulated communication of *data parallelism* or *model parallelism* for this layer can be obtained. Our partitioning scheme ultimately outputs the minimum total communication between two accelerators and a list of parallelism methods we could chose to realize such a minimal communication. Expanding to the partition for an array of accelerators, we use a hierarchical approach which essentially is a recursive function. In each recursion, the minimal communication (*com_h*) at the current hierarchical level is calculated as described above. Then, it calls itself with the input hierarchy levels (h) changed to hierarchy levels ($h-1$) and seeks for the total minimal communication (*com_n*) for the lower hierarchy levels. At last, it returns the total communication *com* by adding the current communication *com_h* and $2 \times$ *com_n*.

To evaluate the effectiveness of the proposed methods, we designed an HMC-based DNN training architecture and tested it with various DNN models from the classic Lenet to VGGs in larger sizes. Our results achieved significant improvements on the performance and energy efficiency. More detailed design explanation and evaluation can be found in [51].

5 DESIGN SPACE EXPLORATION AND OPTIMIZATION

The design of optimized heterogeneous manycore systems for specific application workloads is challenging for a number of reasons. First, the heterogeneity of processing elements (PEs) including CPUs, GPUs, various accelerators, processing-in-memory (PIM) cores, and standard memory results in a very large and complex design space that grows with the system size. Second, we need an optimized network-on-chip (NoC) as the interconnection backbone that can handle the communication requirements of heterogeneous PEs efficiently [25, 26]. For example, in a CPU-GPU based heterogeneous system, CPUs require low memory latency while GPUs need high-throughput data transfers. Moreover, GPUs typically only communicate with a few shared last-level caches (LLCs), which results in many-to-few traffic patterns (i.e., many GPUs communicate with a few LLCs) with negligible inter-GPU communication [11]. This can cause the LLCs to become bandwidth bottlenecks under heavy network loads and lead to significant performance degradation [11]. Third, to design optimized heterogeneous manycore systems, we need to trade-off multiple objectives including power, performance, thermal, and reliability resulting in a complex multi-objective design space exploration (MO-DSE) problem. The design optimization process should also account for the specific characteristics of emerging technologies [36, 37].

Multi-Objective Design Space Exploration Problem. For a fixed system size, we are provided with resources in the form of different types of processing elements (PEs) and communication links. For example, different types of PEs can include CPUs, GPUs, and specialized accelerators. For N PEs and M communication links, we get a fully-specified design space in the form of all possible manycore design configurations $\mathcal{D}$. For example, each manycore design $d \in \mathcal{D}$ corresponds to a specific placement of PEs and communication links. To evaluate each design $d \in \mathcal{D}$, we are given a set of $k > 1$ objectives $O = \{O_1, O_2, \cdots, O_k\}$. Some example objectives include power, performance, and thermal. To come up with practical and feasible designs, we are given a set of physical constraints C. An example constraint is as follows: the overall communication network should have a path between any two PEs.

Our goal is to find the Pareto set (i.e., non-dominated set of designs) $\mathcal{D}^*$ from $\mathcal{D}_C \subseteq \mathcal{D}$, where $\mathcal{D}_C$ is the set of designs that satisfy all the physical constraints C. A design d_2 is dominated by design d_1 if $\forall i, O_i(d_1) \leq O_i(d_2)$; and $\exists j, O_j(d_1) < O_j(d_2)$. Once the Pareto set $\mathcal{D}^*$ is computed, the designer employs some decision-making criteria (e.g., energy-delay-product from simulations) to select the best design d^* from the pareto set $\mathcal{D}^*$. We perform the entire optimization process at the *design-time*. Solving MO-DSE problems for the design of optimized hetergeneous manycore systems poses several challenges: 1) Large design space that grows in size and complexity with increasing system size and diversity of PEs; 2) Hardness and complexity of design-time optimization problem grows with the number of objectives; and 3) Pareto front of designs is non-convex and complex for large number of objectives.

5.1 Guided Design Space Exploration

Common algorithms to solve design-time optimization problems include AMOSA [3] and NSGA-II [17]. Typically, these algorithms execute many unguided and independent searches, from different starting points, to increase the chance of reaching global optima. These algorithms do not leverage the knowledge gained from past design space exploration in an explicit manner and do not allow to leverage the training data from simulations to evaluate the quality of the designs. There is a rich body of literature focusing on solving constrained combinatorial problems (CCPs), especially for the system synthesis [44]. SAT-decoding [40] is the dominant state-of-the-art approach in tackling CCPs. The key idea behind this approach is to use a SAT solver to generate valid solutions in the design space from the genotypes searched over by an evolutionary algorithm. The valid solutions are represented by a set of Pseudo-Boolean (PB) constraints, formulated using binary variables. This approach works very-well in multiple scenarios. However, non-linear constraints cannot be represented by the PB functions formulation used by the SAT-decoding procedure.

Towards the goal of addressing some of the design-time optimization challenges, machine learning (ML) techniques can be used as part of the design optimization framework [31, 32]. Machine-learning techniques can enable the problem-solver (a computational search procedure) to make intelligent search decisions to achieve efficiency for finding (near-) optimal solutions over non-learning-based algorithms. We refer the family of algorithms to solve MO-DSE problems that learn knowledge from designs explored in the past to intelligently explore the design space as *guided design space exploration* framework. In what follows, we list two instances of this framework, namely, MOO-STAGE [27] and MOOS [18].

MOO-STAGE and MOOS algorithms are designed to improve the accuracy of local search based solutions to solve MO-DSE problems. One of the main drawbacks of local search based methods is that the accuracy of solutions produced by them critically depends on the starting designs. Intuitively, if we can select starting designs that will lead the local search procedure to high-quality local optima, it will allow local search algorithms to find (near-) optimal solutions with significantly less number of restarts. ML and data-driven algorithms can be employed to learn search control knowledge from the training data obtained from past local search executions. Subsequently, the learned knowledge can be used to identify the most promising parts of the design space, thereby eliminating a lot of unnecessary exploration.

MOO-STAGE [27] generalizes a machine learning based optimization algorithm called STAGE that was shown to be very effective for single-objective homogeneous manycore design optimization problems [13–15]. The key idea behind MOO-STAGE is to intelligently explore the input design space to improve the efficiency of solving MO-DSE problems. MOO-STAGE is an iterative learning algorithm that leverages the past search experience (Local search) to learn an evaluation function E that can estimate the outcome of performing local search from any given state in the design space (Meta search). The goal of local search (e.g., greedy search) from a given starting design is to traverse through a sequence of neighboring states to find a solution that optimize the given set of objectives O. To accommodate multiple objectives, it employs the Pareto Hyper-Volume (PHV) heuristic to evaluate the quality of a set of solutions. Essentially, the PHV is the size of the objective space that is dominated by a set of solutions. Using the learned evaluation function E, MOO-STAGE intelligently explores the design space by selecting good starting designs for local search. Regression training examples are generated from each design d along the local search trajectory (input is the design d and output is the PHV of local search) and the evaluation function E is induced by giving the aggregate training examples to a supervised learner.

MOOS *multi-objective optimistic search (MOOS)* [18] further improves the scalability and accuracy of solving MO-DSE problems over MOO-STAGE. MOOS performs adaptive design space exploration based on the principle of *optimism in the face of uncertainty* [2] to improve the speed and accuracy of design optimization process. The principle of optimism suggests exploring the most favorable region of the design space based on the experience gained from past exploration. The key insight is that MOOS performs efficient search guided by a data-driven tree-based model over scalarization parameters (small and simple search space) when compared to MOO-STAGE that performs data-driven search guided by learned evaluation function over input design space (large and complex search space). MOOS is an iterative two-stage optimization algorithm. In each iteration, it employs a scalarized objective to select the starting solution for a local search procedure. The parameters of scalarization are chosen adaptively based on a data-driven tree-based model. Local search is performed from the selected starting solution and the tree-based model is updated using the quality of the resulting Pareto set. MOOS was employed to design optimized NoC-enabled homogeneous and heterogeneous manycore systems

[18]. Experimental results demonstrated that MOOS improves the speed of finding solutions similar to state-of-the-art methods (MOO-STAGE and AMOSA) by upto 13X and uncovers solutions that are upto 20% better in terms of NoC. The optimized NoC-enabled 3D manycore systems improve the EDP up to 38% when compared to a 3D mesh-based design optimized for the placement of PEs.

6 DYNAMIC RESOURCE MANAGEMENT

With a rise of machine learning and artificial intelligence, it is common that manycore platforms cannot efficiently execute these algorithms. Different architectures such as GPUs, TPUs, and hardware accelerators are designed and integrated into manycore systems to resolve this issue. The ultimate goal is to maximize resource utilization, performance improvement, and energy efficiency. This requires us to rethink hardware stack as well as software stack to accommodate the increasingly complicated platforms. For the hardware stack, we should combine the benefits of programmability in general-purpose machines, performance efficiency in domain-specific accelerators (DSAs), and parallel computing in GPUs/TPUs. For the software stack, we should consider how to generate tasks in such a way to maximize resource utilization in heterogeneous computing systems. Self-introspection should also be taken into consideration. We should think about how updates and patches of complicates applications can influence hardware design. Therefore, we propose a self-optimizing and self-programming computing system (SOSPCS) framework [55] which provides flexibility, programmability, performance and energy efficiency. It, at compile time, relies on neural network classifiers to generate suitable tasks for different hardware platforms. At run-time, we design reinforcement learning (RL) and imitation learning (IL) based intelligent schedulers to map tasks onto hardware to provide optimal resource utilization and performance speedup.

Compile-time resource allocation via neural networks. Applications have some hidden attributes waiting to be discovered. For example, in neural network algorithms, designers can identify some important tasks such as matrix multiplication and Sigmoid activation function to better implement efficient DSAs. To enable self-optimization, problem formulation is as follows: Given an application graph, find all specialized tasks which can be implemented as DSAs. SOSPCS uses two neural networks to recognize tasks in each application. The first neural network determines the feature type, e.g., matrix multiplication, Sigmoid, or neurons; on the other hand, the second decides the coordinates of each feature, which helps us extract such feature out of the application graph. Once all specialized tasks are identified, we apply some graph partitioning algorithms discussed in Section 3.2 to create tasks with minimal communication cost.

Run-time resource management via reinforcement learning and imitation learning. To enable self-programmability, the next goal is to find a optimal mapping of tasks onto PEs to maximize performance. However, two issues need to be considered: (1) Each task is mapped to its suitable PE set (e.g., FFT tasks should be mapped to FFT accelerators rather than neuron accelerators.). (2) Communication cost among PEs has to be minimal. Therefore, SOSPCS applies a distributed Q-learning algorithm where multiple agents interact with environment independently. These agents, considered as intelligent schedulers, learn to map tasks assigned onto suitable PEs. If one PE is occupied by another task, we perform local search by selecting the available PE nearby to reduce the communication cost. The reward function is designed to prevent agents from assigning wrong tasks to hardware, which could lead to performance degradation.

Recent advances in Imitation Learning (IL) [48] provides an alternative framework to potentially address the above drawbacks of RL. In traditional IL, expert demonstrations are provided as supervised training data (e.g., demonstrations of a human expert driving a car). Then the learner tries to imitate the behavior of the expert in a way that generalizes to similar tasks or situations. These characteristics make IL an exponentially better framework than RL for solving sequential decision-making problems. At a high-level, the difference between IL and RL is same as the difference between supervised learning and exploratory learning. The main caveat of IL is that it assumes the availability of a good Oracle (expert) policy to drive the learning process. Kim et al., [30] constructed an efficient Oracle towards the goal of dynamic power management in voltage frequency island (VFI) based homogeneous manycore systems. Similarly, Mandal et al., [42] constructed efficient Oracle policies for dynamic resource management in heterogeneous mobile platforms. Subsequently, supervised learning is employed to replicate the decisions made by the Oracle policy. Often the supervised learning problem corresponds to learning a classifier or regressor to map states to actions. To learn a robust policy, adavanced IL algorithms such as DAgger can be employed. Due to the nature of sequential decision-making problems, if the control policy makes a mistake, it can cause error propagation and exhibit poor behavior. DAgger mitigates this problem by generating additional training data demonstrating how the Oracle recovers from potential mistakes.

7 CONCLUSIONS AND FUTURE WORK

To enable a paradigm shift in the edge, fig, and exascale computing, we need to develop machine learning and artificial intelligence based approaches to tackle the following challenges:

(1) Analyze the concurrency of algorithms (e.g., degree of parallelization) in relation to the type, size, and dynamics of the data they run on to compensate for hardware features (e.g., deeply-scaled FinFET technologies, near threshold voltage (NTV) induced core slowdown, NTV induced variability of core-to-core throughput).

(2) Evaluate the impact of emerging applications and algorithms (graph analytics, approximate computing algorithms) on the design of AIECS architectures. Communication avoiding linear algebra, randomized numerical linear algebra, and multipole methods have control and data flows that benefit from NoC-based manycore systems. Approximate computing algorithms can self-tune their precision to reduce energy based on mixed precision. Can we build new models of computation for such approximate computing algorithms that allow us to capture variable precision floating point and identify trade-offs between precision and the degree of parallelism?

(3) It is common to update and patch buggy and potentially malicious applications to improve user experience and security. For example, we can view applications as interconnected graphs that evolve over time in terms of structure and characteristics as a function of both needs of end-users and system enhanced functionality requirements. With more updates made on the software

running on autonomous manycore systems, how can we reconstruct a time-varying graph from a partially observed sequence of graphs corresponding to various interacting and evolving application codes? We envision that even with some unknown components hidden in the application graphs, their fractal properties may not change [50, 60, 61] significantly such that we can construct complex multi-layer graph based models.

(4) After reconstruction of the underlying graph based model of dynamic applications, we need to predict the graph structures that will potentially emerge in the future. For example, future work should consider how several updates done in the software running on the self-driving car or the unmanned aerial / underwater vehicles can impact the models of computation and how they influence the design of next-generation efficient hardware platforms. More specifically, how can machine learning and artificial intelligence methods infused in the manycore systems stack predict the upcoming possible updates, patches, and changes in the time varying graphical model of interacting applications in order to dynamically reconfigure the hardware.

ACKNOWLEDGMENTS

Bogdan gratefully acknowledge the support by the National Science Foundation Career award under Grant No. CPS/CNS-1453860, the NSF award under Grant CCF-1837131, the U.S. Army Research Office (ARO) under Grant No. W911NF-17-1-0076, the Okawa Foundation award, and the Defense Advanced Research Projects Agency (DARPA) Young Faculty Award under Grant No. N66001-17-1-4044 support. Doppa was supported in part by the National Science Foundation under Grant OAC-1910213 and Grant IIS- 1845922, and in part by the U.S. Army Research Office under Grant W911NF-17-1-0485 and Grant W911NF-19-1-0162. Li was supported in part by the National Science Foundation under Grant CSR-1717885 and in part by the Air Force Research Lab under Grant FA8750-18-2-0121. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied by the Defense Advanced Research Projects Agency, the Air Force Research Lab, the Department of Defense or the National Science Foundation.

REFERENCES

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 105–117. https://doi.org/10.1145/2749469.2750386

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2-3 (2002), 235–256.

[3] Sanghamitra Bandyopadhyay, Sriparna Saha, Ujjwal Maulik, and Kalyanmoy Deb. 2008. A Simulated Annealing-based Multiobjective Optimization Algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation (TEC)* 12, 3 (2008), 269–283.

[4] Pak K Chan, Martine DF Schlag, and Jason Y Zien. 1994. Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on computer-aided design of integrated circuits and systems* 13, 9 (1994), 1088–1096.

[5] Fan Chen and Hai Li. 2018. EMAT: An Efficient Multi-task Architecture for Trannsfer Learning Using ReRAM. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD '18)*. ACM, New York, NY, USA, Article 33, 6 pages. https://doi.org/10.1145/3240765.3240805

[6] F. Chen, L. Song, and Y. Chen. 2018. ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 178–183. https://doi.org/10.1109/ASPDAC.2018.8297302

[7] Fan Chen, Linghao Song, Hai Helen Li, and Yiran Chen. 2019. ZARA: A Novel Zero-free Dataflow Accelerator for Generative Adversarial Networks in 3D ReRAM. In *Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19)*. ACM, New York, NY, USA, Article 133, 6 pages. https://doi.org/10.1145/3316781.3317936

[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967

[9] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[10] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 27–39. https://doi.org/10.1109/ISCA.2016.13

[11] Wonje Choi, Karthi Duraisamy, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. On-Chip Communication Network for Efficient Training of Deep Convolutional Networks on Heterogeneous Manycore Systems. *IEEE Transactions on Computers (TC)* 67, 5 (2018), 672–686.

[12] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An industry-standard API for shared-memory programming. *Computing in Science & Engineering* 1 (1998), 46–55.

[13] Sourav Das, Janardhan Rao Doppa, Daehyun Kim, Partha Pratim Pande, and Krishnendu Chakrabarty. 2015. Optimizing 3D NoC Design for Energy Efficiency: A Machine Learning Approach. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 705–712.

[14] Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2017. Design-space Exploration and Optimization of an Energy-efficient and Reliable 3-D Small-world Network-on-Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36, 5 (2017), 719–732.

[15] Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2017. Monolithic 3D-Enabled High Performance and Energy Efficient Network-on-Chip. In *Proceedings of IEEE International Conference on Computer Design (ICCD)*. 233–240.

[16] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1223–1231. http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf

[17] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation (TEC)* 6, 2 (2002), 182–197.

[18] Aryan Deshwal, Nitthilan Kannappan Jayakodi, Biresh Kumar Joardar, Janardhan Rao Doppa, and Partha Pratim Pande. 2019. MOOS: A Multi-Objective Design Space Exploration and Optimization Framework for NoC enabled Manycore Systems. *ACM Transactions on Embedded Computing Systems (TECS)* (2019).

[19] Nikil Dutt, Axel Jantsch, and Santanu Sarma. 2016. Toward smart embedded systems: A self-aware system-on-chip (soc) perspective. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 2 (2016), 22.

[20] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 283–295. https://doi.org/10.1109/HPCA.2015.7056040

[21] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[22] Miao Hu, Hai Li, Qing Wu, and Garrett S. Rose. 2012. Hardware Realization of BSB Recall Function Using Memristor Crossbar Arrays. In *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*. ACM, New York, NY, USA, 498–503. https://doi.org/10.1145/2228360.2228448

[23] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2897937.2898010

[24] Axel Jantsch, Nikil Dutt, and Amir M Rahmani. 2017. Self-awareness in systems on chipâĂŤA survey. *IEEE Design & Test* 34, 6 (2017), 8–26.

[25] Biresh Kumar Joardar, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. Hybrid on-chip communication architectures for heterogeneous manycore systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. ACM, 62.

[26] Biresh Kumar Joardar, Ryan Gary Kim, Janardhan Rao Doppa, and Partha Pratim Pande. 2019. Design and Optimization of Heterogeneous Manycore Systems Enabled by Emerging Interconnect Technologies: Promises and Challenges. In *Proceedings of IEEE/ACM International Conference on Design, Automation & Test in Europe Conference & Exhibition, (DATE)*. 138–143.

[27] Biresh Kumar Joardar, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. Learning-based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems. *IEEE Trans. Comput.* 68, 6 (2018), 852–866.

[28] Biresh Kumar Joardar, Bing Li, Janardhan Rao Doppa, Hai Li, Partha Pratim Pande, and Krishnendu Chakrabarty. 2019. REGENT: A Heterogeneous ReRAM/GPU-based Architecture Enabled by NoC for Training CNNs. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*. 522–527.

[29] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a Warehouse-scale Computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 158–169. https://doi.org/10.1145/2749469.2750392

[30] Ryan Gary Kim, Wonje Choi, Zhuo Chen, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2017. Imitation Learning for Dynamic VFI Control in Large-Scale Manycore Systems. *IEEE Transactions on VLSI Systems (TVLSI)* 25, 9 (2017), 2458–2471.

[31] Ryan Gary Kim, Janardhan Rao Doppa, and Partha Pratim Pande. 2018. Machine learning for design space exploration and optimization of manycore systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. IEEE, 48.

[32] Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. Machine Learning and Manycore Systems Design: A Serendipitous Symbiosis. *IEEE Computer* 51, 7 (2018), 66–77.

[33] S. Kim, M. Ishii, S. Lewis, T. Perri, M. BrightSky, W. Kim, R. Jordan, G. W. Burr, N. Sosa, A. Ray, J. . Han, C. Miller, K. Hosokawa, and C. Lam. 2015. NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning. In *2015 IEEE International Electron Devices Meeting (IEDM)*. 17.1.1–17.1.4. https://doi.org/10.1109/IEDM.2015.7409716

[34] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv e-prints*, Article arXiv:1404.5997 (Apr 2014), arXiv:1404.5997 pages. arXiv:cs.NE/1404.5997

[35] Vipin Kumar. 2002. *Introduction to parallel computing*. Addison-Wesley Longman Publishing Co., Inc.

[36] Dongjin Lee, Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2018. Performance and Thermal Tradeoffs for Energy-Efficient Monolithic 3D Network-on-Chip. *ACM Trans. Design Autom. Electr. Syst. (TODAES)* 23, 5 (2018), 60:1–60:25.

[37] Dongjin Lee, Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2019. Impact of Electrostatic Coupling on Monolithic 3D-enabled Network on Chip. *ACM Trans. Design Autom. Electr. Syst. (TODAES)* (2019).

[38] Mu Li, David G. Andersen, Alexander Smola, and Kai Yu. 2014. Communication Efficient Distributed Machine Learning with the Parameter Server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 19–27. http://dl.acm.org/citation.cfm?id=2968826.2968829

[39] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Yu Wang, Hao Jiang, M. Barnell, Qing Wu, and Jianhua Yang. 2015. RENO: A high-efficient reconfigurable neuromorphic computing accelerator design. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2744769.2744900

[40] Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jurgen Teich. 2007. Sat-Decoding in Evolutionary Algorithms for Discrete constrained Optimization problems. In *2007 IEEE Congress on Evolutionary Computation*. IEEE, 935–942.

[41] Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen. 2017. DaDianNao: A Neural Network Supercomputer. *IEEE Trans. Comput.* 66, 1 (Jan. 2017), 73–88. https://doi.org/10.1109/TC.2016.2574353

[42] Sumit Mandal, Ganapati Bhatt, Chetan Arvid Patel, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Ogras. 2019. Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning. *IEEE Transactions on VLSI Systems (TVLSI)* (2019).

[43] D. Pandiyan and C. Wu. 2014. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 171–180. https://doi.org/10.1109/IISWC.2014.6983056

[44] Jacopo Panerati, Donatella Sciuto, and Giovanni Beltrame. 2017. Optimization Strategies in Design Space Exploration. In *Handbook of Hardware/Software Codesign*. 189–216.

[45] Jürgo S Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid, and Emine Calis. 2015. The benefits of self-awareness and attention in fog and mist computing. *Computer* 48, 7 (2015), 37–45.

[46] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. 2009. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In *2009 17th Euromicro international conference on parallel, distributed and network-based processing*. IEEE, 427–436.

[47] Shankar Ganesh Ramasubramanian, Rangharajan Venkatesan, Mrigank Sharad, Kaushik Roy, and Anand Raghunathan. 2014. SPINDLE: SPINtronic Deep Learning Engine for Large-scale Neuromorphic Computing. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design (ISLPED '14)*. ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/2627369.2627625

[48] Stefan Schaal. 1999. Is Imitation Learning The Route To Humanoid Robots? *Trends in cognitive sciences* 3, 6 (1999), 233–242.

[49] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 14–26. https://doi.org/10.1109/ISCA.2016.12

[50] Jayson Sia, Edmond Jonckheere, and Paul Bogdan. 2019. Ollivier-Ricci Curvature-Based Method to Community Detection in Complex Networks. *Scientific reports* 9, 1 (2019), 9800.

[51] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen. 2019. HyPar: Towards Hybrid Parallelism for Deep Learning Accelerator Array. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 56–68. https://doi.org/10.1109/HPCA.2019.00027

[52] L. Song, X. Qian, H. Li, and Y. Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 541–552. https://doi.org/10.1109/HPCA.2017.55

[53] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen. 2018. GraphR: Accelerating Graph Processing Using ReRAM. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 531–543. https://doi.org/10.1109/HPCA.2018.00052

[54] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2018. Prometheus: Processing-in-memory heterogeneous architecture design from a multi-layer network theoretic strategy. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1387–1392.

[55] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2019. Self-Optimizing and Self-Programming Computing Systems: A Combined Compiler, Complex Networks, and Machine Learning Approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 6 (2019), 1416–1427.

[56] Yao Xiao, Yuankun Xue, Shahin Nazarian, and Paul Bogdan. 2017. A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 217–224.

[57] Yuankun Xue and Paul Bogdan. 2015. User cooperation network coding approach for NoC performance improvement. In *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM, 17.

[58] Yuankun Xue and Paul Bogdan. 2016. Improving NoC performance under spatio-temporal variability by runtime reconfiguration: a general mathematical framework. In *2016 tenth IEEE/ACM international symposium on networks-on-chip (NOCS)*. IEEE, 1–8.

[59] Yuankun Xue and Paul Bogdan. 2016. Scalable and realistic benchmark synthesis for efficient NoC performance evaluation: A complex network analysis approach. In *2016 international conference on hardware/software codesign and system synthesis (CODES+ISSS)*. IEEE, 1–10.

[60] Yuankun Xue and Paul Bogdan. 2017. Reliable multi-fractal characterization of weighted complex networks: algorithms and implications. *Scientific reports* 7, 1 (2017), 7487.

[61] Yuankun Xue and Paul Bogdan. 2019. Reconstructing missing complex networks against adversarial interventions. *Nature communications* 10, 1 (2019), 1738.

[62] Yuankun Xue, Ji Li, Shahin Nazarian, and Paul Bogdan. 2017. Fundamental challenges toward making the iot a reachable reality: A model-centric investigation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 22, 3 (2017), 53.

[63] Bonan Yan, Qing Yang, Wei-hao Chen, Kung-tang Chang, Jian-wei Su, Chien-hua Hsu, Sih-han Li, Precision-configurable In Situ, Nonlinear Activation, and Tsing Hua. 2019. RRAM-based Spiking Nonvolatile Computing-In-Memory Processing Engine with Precision-Configurable In Situ Nonlinear Activation. *IEEE Symposium on VLSI Technology* (2019), T86–T87.

[64] H. Yu, Y. Wang, S. Chen, W. Fei, C. Weng, J. Zhao, and Z. Wei. 2014. Energy efficient in-memory machine learning for data intensive image-processing by non-volatile domain-wall memory. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 191–196. https://doi.org/10.1109/ASPDAC.2014.6742888