nZESPA: A Near-3D-Memory Zero Skipping Parallel Accelerator for CNNs

Palash Das, *Member, IEEE*, and Hemangee K. Kapoor, *Senior Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) are one of the most popular machine learning tools for computer vision. The ubiquitous use in several applications with its high computation-cost has made it lucrative for optimization through accelerated architecture. State-of-the-art has either exploited the parallelism of CNNs, or eliminated computations through sparsity or used near-memory processing (NMP) to accelerate the CNNs. We introduce NMP-fully sparse architecture, which acquires all three capabilities. The proposed architecture is parallel and hence processes the independent CNN tasks concurrently. To exploit the sparsity, the proposed system employs a dataflow, namely, *Near-3D-Memory Zero Skipping Parallel dataflow* or *nZESPA* dataflow. This dataflow maintains the compressed-sparse encoding of data that skips all ineffectual zero-valued computations of CNNs. We design a custom accelerator which employs the nZESPA dataflow. The grids of nZESPA modules are integrated into the logic layer of the hybrid memory cube. This integration saves a significant amount of off-chip communications while implementing the concept of NMP. We compare the proposed architecture with three other architectures which either do not exploit sparsity (NMP-dense) or do not employ NMP (traditional-fully sparse) or do not include both (traditional-dense). The proposed system outperforms the baselines in terms of performance and energy consumption while executing CNN inference.**

***Index Terms*—Accelerator architecture, convolutional neural network (CNN), near-memory processing (NMP), sparsity.**

I. Introduction

CONVOLUTIONAL neural network (CNN) has emerged as one of the most popular tools in the domain of deep learning to solve complex problems. A typical CNN algorithm includes two phases: 1) training and 2) inference. In the training phase, new parameters are iteratively learned by the models. Upon completion of the training, the inference phase starts, and the performance of the trained model on the unseen data is assessed based on the obtained accuracy. Toward achieving the accuracy of beyond human perception, the CNN models have grown deeper (with several hidden layers) and exceptionally large in size. The number of both neurons and synaptic weights of the CNNs has increased dramatically (neurons: kilo-million neurons and synaptic weights: up to 10 billion) [1], [2]. CNNs require ten to a hundred megabytes of parameters on billions of operations in a single inference pass. Consequently, an extensive amount of computations and memory accesses are involved in CNN inference operations, particularly with the emergence of deep networks and larger input sets like high-definition videos [3], [4]. These compute and data-intensive CNN operations paralyze the memory hierarchy and increase the costly off-chip DRAM accesses, which require higher cycles and energy to transfer data between the last-level cache and main memory. Eventually, the system's throughput in terms of performance and energy consumption degrades. Any optimization of the CNN algorithm regarding its performance and energy consumption benefits varieties of applications that include CNNs as the core computation module. Toward optimizing the CNN algorithm, researchers have exploited the inherent parallelism of the CNN operations [5]–[7]. A few groups have eliminated unnecessary computations by utilizing the abundant sparsity present in the neurons (dynamic) [8], [9] and synaptic weights (static) [10], [11]. SCNN [12] and EIE [13] have exploited both static and dynamic sparsity to reduce the number of computations of the CNNs. Researchers have also put in efforts to reduce the costly off-chip main memory accesses by near-memory processing (NMP) to achieve a high system's throughput while accelerating the CNNs [14]–[16].

In the first category, the parallelism of the CNN operations has been exploited by the use of multiple processing units, resulting in an improved systems throughput in terms of performance and energy consumption. In the second category, people have utilized the sparsity property to reduce the computations of CNNs, thereby improving the systems throughput. Sparsity can be defined as the number of zeros present in the weight (synapse) and activation (neuron) matrices [17]. This sparsity is usually static in nature for the weights as it stems out by pruning connections in the network during the training. In contrast, the sparsity in activations, raised by the thresholding of the rectified linear unit (ReLU), is data-dependent and hence dynamic in nature. Multiplications and additions of these weights and activations are the primitive operations for the inference. Since multiplication or addition with zero produces nothing, these operations can be eliminated by over an order of magnitude to skip the costly computations. Around 18%–85% static and 25%–60% dynamic sparsity is observed for the different layers of CNN benchmarks in [17]. In [12], a similar observation is made (static: 20%–80%, dynamic: 50%–70%). This has motivated many researchers to exploit the sparsity property. In some research, only the one-sided sparsity, like the static or the dynamic sparsity, has been investigated. Others have explored the benefits of both static

Manuscript received April 9, 2020; revised July 8, 2020; accepted August 24, 2020. Date of publication September 8, 2020; date of current version July 19, 2021. This article was recommended by Associate Editor H. Li. *(Corresponding author: Palash Das.)*

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati 781039, India (e-mail: palash.das@iitg.ac.in; hemangee@iitg.ac.in).

Digital Object Identifier 10.1109/TCAD.2020.3022330

and dynamic sparsity, and thus fully sparse. Nonetheless, the above two approaches still fail to address the issue of costly main memory accesses while fetching/storing data in the main memory. One solution to this problem is NMP, where some computations are offloaded to the processing units, placed near to the memory. The NMP approach helps in amortizing the off-chip accesses. Several recent projects on the NMP of neural networks have manifested that, the systems having NMP capability expedite the performance while being significantly energy efficient. As explained above, the existing works have been able to leverage isolated benefits of parallelism or sparsity or NMP by their proposed architecture. Addressing all three issues by one novel architecture is still a problem and has not been implemented.

In this article, our proposed system exploits the parallelism by distributing the CNN tasks among the proposed processing units, placed near to the memory. In this context, we choose 3-D memory, specifically hybrid memory cube (HMC) [18], for the near-memory integration of the designed processing units. We utilize both static and dynamic sparsity to eliminate the costly multiply accumulate (MAC) operations of the CNN inference without affecting the accuracy. Apart from being near-memory, we use data compression on both weights and activations. Encoding of sparse weights and activations provides twofold benefits: 1) it reduces the number of DRAM access from the HMC's logic layer (LoB) where we place the proposed hardware and 2) it helps to accommodate the required weights and activations in a comparatively smaller sized buffer in the proposed hardware. Finally, we have successfully curtailed the off-chip main memory accesses using the NMP approach while executing the inference. The reason is, the majority of the computations on the data take place inside the LoB of the HMC device without bringing them to the conventional cache hierarchy. Toward achieving these goals, we design our proposed hardware, namely, near-3D-memory zero skipping parallel accelerator (nZESPA), and integrate multiple such hardware units in the LoB of the HMC device. The proposed system outperforms the baselines in terms of performance and energy consumption, as shown in Section VI-C. The salient contributions are as follows.

1) To harness parallelism, we design an efficient sparsity-aware near-3D-memory dataflow for layer wise processing of data, namely, nZESPA.
2) We design specialized hardware (nZESPA) and integrate a grid of those units with the individual vaults (explained in Section IV) in the LoB of the HMC device. The nZESPA is area efficient and can be easily integrated into the existing ecosystem of the HMC.
3) We exploit static and dynamic sparsity, which reduces the computation cost by eliminating the zero-valued computations. The sparse data is also represented and utilized in a compressed form reducing the buffer requirement in the implemented hardware.
4) To quantify the ability of the proposed system having fully sparse CNN accelerator with NMP (NMP-fully sparse), we compared it with three other variants of architectures: ⓐ System having dense CNN accelerator without NMP (traditional-dense); ⓑ System having fully sparsed CNN accelerator without NMP (traditional-fully sparse); and ⓒ System having dense CNN accelerator with NMP (NMP-dense). The NMP-fully sparse outperforms the other three architectures in terms of performance and energy consumption.
5) We also compare the proposed NMP-fully sparse architecture with state-of-the-art approaches exploring either static or only dynamic sparsity.
6) The throughput of the proposed system is evaluated with three widely used state-of-the-art ConvNets/networks with different shapes and sizes: AlexNet [1], VGG-16 [19], and ResNet-34 [20].

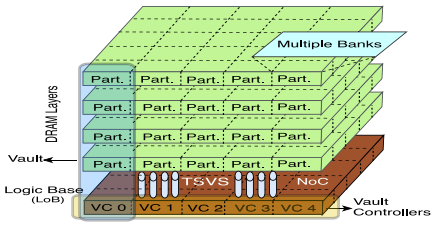


Fig. 1. Conceptual view of HMC.

II. Background and Motivation

A. Hybrid Memory Cube

Fig. 1 depicts the abstract architecture of the HMC device. Generally, HMC [18] devices contain 4 to 8 dies of DRAM cell arrays or DRAM layers. A 3-D-memory die is formed by stacking DRAM layers on top of each other. Each DRAM layer includes multiple DRAM banks. The high-speed through-silicon-vias (TSVs) are integrated into the HMC module to transfer data to/from the DRAM layers. A separate dedicated layer (LoB), containing all the interconnects and controllers, are placed at the bottom of the memory layers. This layer helps to fetch and store data in the memory die. Each of the memory layers is divided into 16 to 32 partitions (Part.), which are the collection of the memory banks. An individual stack of these partitions constitutes a vault (vertical slice of the memory die, as indicated in Fig. 1). Each vault is controlled by a separate memory controller, namely, the vault controller (VC). All the memory references, corresponding to the individual vaults, are managed independently by these VCs. HMC uses the host memory controller and SerDes links for off-stack communications. The advantage of using 3D-memory like HMC is its high bandwidth (160–250 Gb/s) with 3 to 5 times lower access energy compared to DDR3 [18].

B. CNN Fundamentals

As mentioned before, CNNs comprise of two phases: 1) training and 2) inference. While the training phase consists of three key steps *viz.*: 1) forward propagation; 2) backpropagation; and 3) weight gradient update, the inference has one phase, i.e., forward propagation. The four most common layers of the forward propagation are convolution (CONV), pooling, normalization, and fully connected. The proposed nZESPA primarily aims to optimize the computations of the CONV layers as they (CONV layers) are the core building block of any

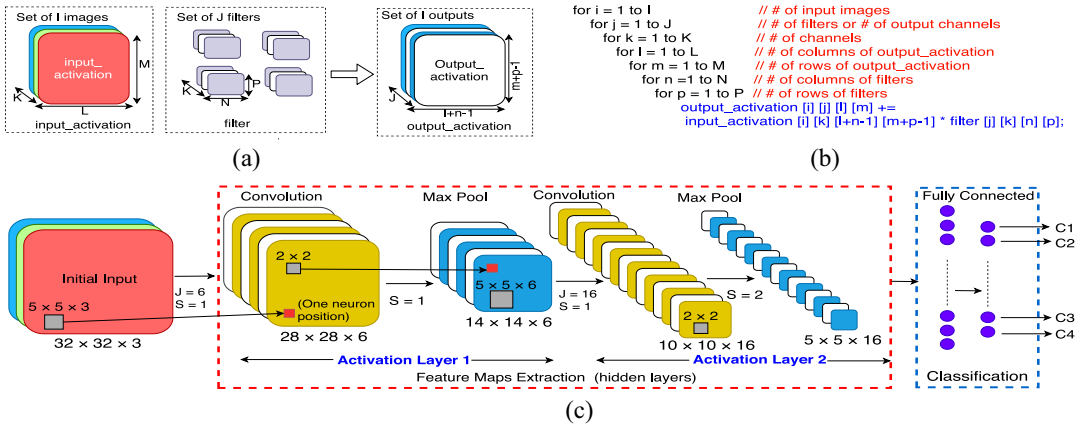


Fig. 2. Architecture of CNN. (a) Parameters of CNN inference. (b) 7-D CNN deep-nested loop. (c) CNN inference/forward propagation.

CNN and computationally heavier than the rest [21], [22]. The architecture of the feed-forward inference is shown in Fig. 2. The algorithm starts extracting basic features in the initial layers, and gradually complicated features are extracted in the subsequent layers. This algorithm takes two types of matrices as input: 1) input_activation and 2) filter matrices (kernels). Convolutions are the specialized kind of linear operations that can be reduced to matrix multiplications only after performing the lowering or image-to-column transformations of the activation matrices. The hidden layers of a CNN customarily consist of a series of convolutional layers that convolve with multiplications or dot products of two matrices (input_activations and kernels) at each neuron position (shown in Fig. 2). The size of the feature maps is reduced by the subsequent pooling layers. Output activations (OAs) of one layer are then fed as input activations to the next layer. A new set of kernels representing more sophisticated features is then applied for the CONV operations. After the operations of all the layers, the inference phase terminates by producing the prediction results with satisfactory accuracy.

The complete set of operations of the activation layers is represented by a 7-D deep-nested loop consisting of $\langle i, j, k, l, m, n, p \rangle$ variables shown in Fig. 2(b). Interestingly, this sequence of loops of $\langle l, m, n, p \rangle$ variables are independent and can be reordered since multiply add operations are associative. This helps to achieve parallelism by distributing the computation among various processing modules. The way of distributing the computations and data to be processed creates different dataflows, which are one of the major differentiators of the neural network acceleration works. The reason is; different dataflows lead to different designs, having significant impacts on the system's performance and energy consumption. The meaning of the loop variables ($\langle i, j, k, l, m, n, p \rangle$) is also explained in Fig. 2(b).

C. Motivations

The term sparsity can be defined as the fraction of zero values found in the layer's activation and weight matrices. The abundant sparsity, present in the CNNs, brings enormous opportunities for the system architects to design highly

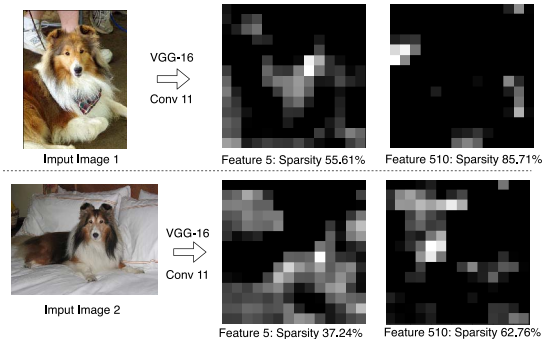


Fig. 3. Dynamic nature of the activation-sparsity.

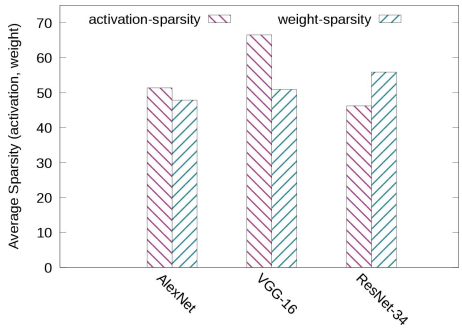


Fig. 4. Exploitable sparsity for the ConvNets.

efficient systems with optimized energy consumption. In practice, all the widely used ConvNets/networks include sparsity in both the activation and weight matrices. The effective approach to create weight sparsity is to prune the network during the training phase. One of the effective techniques to prune the weights is shown in [23]. Here, the weight values which are closed to zero (e.g., within a certain threshold) are set to zeros. This pruning process affects the accuracy. To regain accuracy, they retrain the network. These two pruning and retraining steps result in a smaller network with accuracy, very close to the original network. Once the training is done, the weight sparsity remains unchanged, and hence static in nature.

Activation sparsity is created primarily by the nonlinear operator like ReLU as it forces all negative activation values to be clamped to zeros. The sparsity of activations is highly

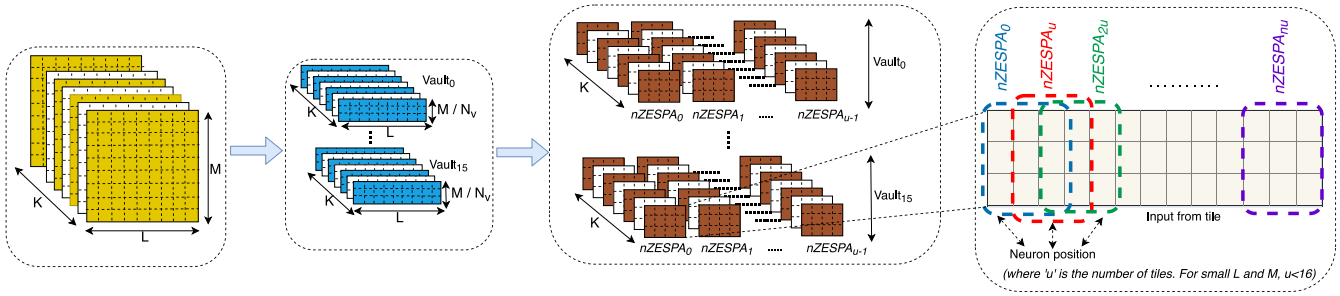


Fig. 5. Data distribution for nZESPA dataflow. (a) Entire data set. (b) Data blocks. (c) Data tiles. (d) Load distribution on nZESPA grid (where "u" is the number of tiles. For small L and M, $u < 16$).

data-dependent and hence dynamic in nature. Fig. 3 shows the dynamic nature of the activation sparsity. The CONV11 layer of VGG-16 is shown for the two different input images from the ImageNet dataset. A variance in sparsity is observed across the input images. Note that, here, the black pixels represent the zero values that are responsible for the sparsity. To inspect the activation and weight sparsity across the hidden layers of all three networks, we use the PyTorch framework [24] along with the pruning algorithm of [23]. Fig. 4 illustrates the average sparsity (in percentage) in the activation and weight matrices of all three networks that have been used for evaluation. We observed a range of ∼20%–70% weight and ∼28%–80% activation sparsity across the different layers of the networks. For this work, we assume a batch size of 1, which is common for inferencing tasks.

III. nZESPA Dataflow

The *Near-3D-Memory Zero Skipping Parallel* dataflow or *nZESPA* dataflow is specifically designed to maintain the streaming of data through TSVs between the DRAM layers of the memory die and the nZESPA modules, placed on the LoB of HMC device. The objectives of the dataflow are multidimensional while executing the inference. The *Near-3D-Memory* term represents its active workspace, which is the logic-layer (LoB), close to the memory die. The *Zero Skipping* term is used as it is specially designed to perform its computations on the compressed form of data (activations + weights) obtained after removing zeroes. The compressed data helps in removing all MAC operations associated with zero-valued weights and activations, leading to the exploitation of sparsity property. The nZESPA dataflow keeps both the activations and weight model in the vaults until the end of the entire algorithm. This property increases the data reuse to some extent and reduces the overhead of data-partitioning after each activation layer. To harness the parallelism, the weight matrices are replicated in the vaults while the data is partitioned (shown in Fig. 5) across the vaults for the parallel processing by the nZESPA grids. The dataflow provides parallelism in the form of intra and inter vault parallelism, as explained in the subsequent section. Since we integrate the nZESPA grids close to memory (NMP), the data can be transferred between the DRAM layers and nZESPA modules with negligible transmission overhead [14]. Consequently, we can exchange the activations and the filters between the DRAM layers and the local buffers of nZESPA modules with minimal effect on the system's throughput. One inefficiency that stems out during the parallel execution of the inference tasks is load imbalance. Load imbalance often occurs as an effect of data reuse scheme. Like in [12], the filter elements can be broadcast among the processing elements (PEs) for performing the MAC operations while keeping the input_activations stationary in the local buffers of PEs. The input maps having different sparsity, sparser maps finish early than denser maps, leading to the under-utilization of resources due to the load imbalance. Alternatively, holding the filter elements and broadcasting the input_activations incurs the same problem. The load imbalance can be alleviated if the PEs independently fetch the weights and activations to the local buffers by sacrificing the data reuse. Thus, there is a fundamental reuse-imbalance tradeoff while designing the dataflow. We fetch the activations and filters in the local buffers instead of broadcasting them among the nZESPA modules as a solution to the load imbalance problem.

A. Data Partitioning for nZESPA

Fig. 5 explains the data distribution and its granularity in more detail. The entire $L \times M \times K$ amount of activation inputs [Fig. 5(a)] are chopped horizontally, leading to data blocks, as shown in Fig. 5(b). Here, L and M are the counts of columns and rows, respectively, for each input channel; and the total number of channels is represented by K (same as shown in Fig. 2). Each block of $L \times (M/N_v) \times K$ elements is distributed across the N_v number of vaults of the HMC. We choose the 16-vault configuration of the HMC 2.0 for our experiment. However, one can choose the other available HMC configurations as in HMC 2.1. It (2.1 version) has 32 vaults, which can be used for the integration of 32 nZESPA grids. Note that our design of nZESPA modules are independent of memory architecture and can be adopted in any 3D-stacked memory with minimal changes in the existing ecosystem. Each block in the respective vaults is logically divided into smaller pieces of tiles [Fig. 5(c)], each having $L/N_{spc} \times (M/N_v) \times K$ elements. Here, N_{spc} represents the number of nZESPA modules (16 in proposed NMP-fully sparse), integrated into each vault. Fig. 5(d) shows the load distribution in a finer granularity when L, M is small, and the number of tiles becomes less than N_{spc}. To maximize hardware utilization for smaller layers (small L

and M), the nZESPA modules within a grid perform the convolutions simultaneously in the sequential neuron positions of a tile (based on the stride value available in register-stack), as shown in Fig. 5(d). However, for larger layers (large L, M) where the number of tiles is at least equals to N_{spc}, the nZESPA modules of a grid perform convolution simultaneously on multiple tiles, as shown in Fig. 5(c). This approach reduces the number of idle cycles of nZESPA modules.

B. Intra and Inter Vault Parallelism of nZESPA

The nZESPA dataflow provides twofold parallelism. The intravault parallelism is achieved within a single vault. In NMP-fully sparse architecture, we placed a grid of 16 nZESPA modules in each vault of the HMC device. Each nZESPA module within a vault can perform the MAC operations on the corresponding tile [Fig. 5(c)] independently in parallel with others. This concurrent processing of the tiles leads to intravault parallelism. The intravault parallelism is exploited on a single block of data [Fig. 5(b)]. Further, as shown in Fig. 5(d), the nZESPA modules can compute on different sections of data of a tile in parallel, leading to a reduced number of idle cycles for the smaller layers. HMC, having multiple vaults, provides a suitable environment to implement a highly parallel system. We integrate the nZESPA grids with all the vaults. These nZESPA grids parallelly process all the data blocks [Fig. 5(b)], resulting in intervault parallelism. Note that our architectural setup includes 16-vault HMC configurations with 16 nZESPA modules (1 grid) in each vault to leverage intra and inter vault parallelism. Increasing the number of grids by increasing the number of vaults as in HMC 2.1 can increase the performance by additional intervault parallelism in exchange for additional power consumption by the system. Similarly, increasing the number of nZESPA modules in a single grid can increase the intravault parallelism while also incurring additional overhead on area and power, leading to design tradeoffs for the system designers.

Due to the sliding window nature of the convolutions along with the horizontal and vertical partitioning scheme of nZESPA dataflow, there will be some missing input activations at the edges of each block and tile (data blocks and tiles are shown in Fig. 5), resulting in partial results at some neuron positions. To address the cross tile dependencies, we replicate additional input activations in the activation buffers (ABs) of each nZESPA module. We also replicate additional activations in each vault for the edges of the data blocks. While the former solves the cross tile dependency problem for intravault parallelism, the latter solves the issue for intervault parallelism.

C. Data Compression

The sparsity of both the weights and activations is harnessed by the support of a compressed data format. Fig. 6 represents the compression technique used in this nZESPA dataflow. N, P, and K have the same meaning, as shown in Fig. 2. We adopted a variant of the previously proposed compression algorithm [13], [25]. The compressed format of data assists the nZESPA dataflow in eliminating the zero-valued computations associated with both the weights and activations.

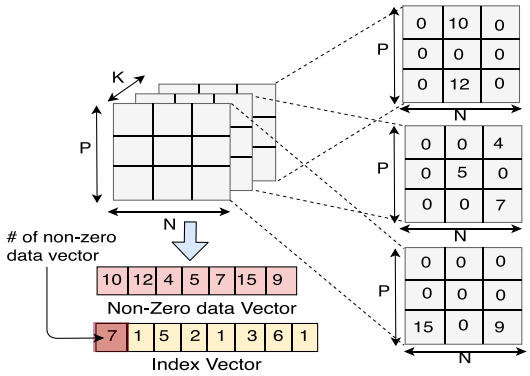


Fig. 6. Illustration of compression technique.

In Fig. 6, we have shown the compression technique for a single weight matrix. The same technique is followed for the compression of all the weight and activation matrices. From Fig. 6, it can be seen that the number of $K \times P \times N$ (where $K = P = N = 3$) filter elements (weights) of a filter reduced to the number of nonzero elements (7 in this case) which are stored in nonzero data vector. The first entry (shown in red in Fig. 6) of the index vector stores the number of nonzero elements of a filter/activation matrix, and it is followed by the number of zeros before each nonzero element. We considered 4 bit/entry for the index vector, which allows up to 15 consecutive zeros between any two nonzero elements. If the two nonzero elements are further apart from each other, then a zero-value place holder can be inserted without any notable loss in the compression efficiency. As the dimensions of the filters and input activation layers are known, we can decrypt the compressed data with the help of the above two vectors.

The nZESPA controller is designed to perform the inference operations on this specific format of data. At each neuron position (as shown in Fig. 2) for the OAs, the nZESPA controller only fetches nonzero input activations from the nonzero data vector. Upon receiving the nonzero input activations, the nZESPA controller checks their validity for the corresponding neuron position with the help of the index vector. For the valid nonzero activations, the controller also searches the corresponding nonzero weights. Upon a successful search, the activations and their corresponding weights are multiplied and accumulated. Consequently, all zero-valued computations are skipped as only nonzero elements participate in the MAC operations.

IV. System Architecture

The architecture, shown in Fig. 7, is an abstract view of the proposed NMP-fully sparse architecture. The system consists of a host processor and a two-level cache hierarchy (L1 split cache and shared L2). Instead of traditional DDRx memory, we choose Micron's HMC as the main memory for NMP. The reason is: 3-D memories are the promising technology for near-memory integration of logic. We can easily integrate the custom accelerators in the logic layer (LoB) of the device without sacrificing the expensive memory area. Additionally, it provides an order of magnitude higher bandwidth (160 to

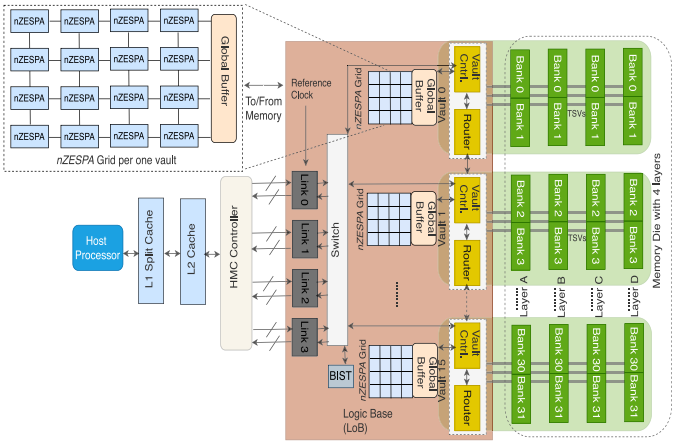


Fig. 7. Proposed NMP-fully sparse architecture.

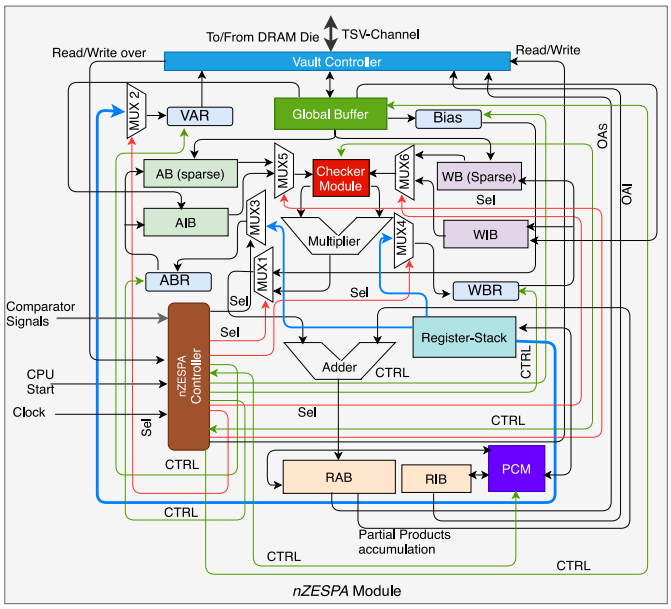


Fig. 8. Abstract view of nZESPA architecture.

250 Gb/s) with 3 to 5 times lower access energy than conventional DDRx memory [18]. The on-chip HMC controller manages all the global memory references. The HMC, considered in our setup, consists of four memory layers (A, B, C, and D), each having a size of 4-Gbit, which sum up to 2 GB of memory in total. The two banks from each layer constitute a vault that has a separate VC. The VCs work independently and handle all the memory references local to the corresponding vault. VCs exchange the nonlocal data with the other vaults by the help of routers, connected through a 2-D mesh network-on-chip (NoC). The detailed NoC circuitry is omitted in Fig. 7 for simplicity. The LoB layer of the original HMC device primarily consists of VCs, a crossbar network to interconnect the vaults, the off-chip link SerDes, and other testing circuitry [18]. To integrate the grid of nZESPA, we replaced the crossbar network with the 2-D mesh NoC, similar to the previous studies [26], [27].

16 nZESPA modules are connected through NoC to form a grid, as shown in the top-left part of Fig. 7. Each grid is connected to the individual VCs. The logic die of HMC device that we have considered permits approximately 50–60 mm^2 in total or 3.5 mm^2 per vault of additional space for near-memory logic apart from the existing circuitry [18]. This clearly manifests that there is a strict area budget for a larger array of logic. For our experiment, we placed a grid of 16 nZESPA modules in each vault as it easily fits within the available space of 3.5 mm^2 in each vault. On UMC 90-nm technology, the entire grid of 16 such nZESPA modules incurs an area overhead of 2.56 mm^2, which is lesser than the area budget of 3.5 mm^2. On the other hand, it is difficult to integrate larger logic arrays in the logic layer of 3-D memory. For example, state-of-the-art like [8] incurs an area overhead of 16 mm^2 on TSMC 65-nm technology for larger PEs array, thus making such designs suitable for on-chip rather than near-memory integration.

In addition to the traditional cache hierarchy, the proposed NMP-fully sparse architecture includes local buffers (weight and AB) in the nZESPA modules, a shared global buffer for the nZESPA grid, and DRAM banks. These buffers are incorporated since accessing the data from DRAM memory die is costlier than the rest of the two. The NMP-fully sparse architecture discussed above, delivers threefold benefits: 1) the concurrent processing of the inference operations through intra and inter vault parallelism; 2) exploitation of both static and dynamic sparsity through nZESPA dataflow on the compressed data; and 3) faster memory accesses due to the NMP capability. Toward achieving these features, we have proposed the nZESPA dataflow and designed the nZESPA hardware that resembles the nZESPA dataflow. The proposed nZESPA module is synthesized in hardware using the Cadence tool-set (described in Section VI-B1).

V. nZESPA Architecture and Operations

The complete microarchitecture of nZESPA, being complex and difficult to represent by a diagram, we have presented an abstract overview of the designed unit in Fig. 8.

1) The nZESPA controller, being the master of the entire circuitry, drives the whole module. The nZESPA controller fetches all the required data of a tile (shown in Fig. 5) from the DRAM die through the TSVs with the help of the VC. The global buffer within the individual vaults is shared among the nZESPA modules of a grid belonging to that vault. The data are initially brought into the global buffer and then transferred to the respective buffers with the help of the nZESPA controller. The vault address register (VAR) holds the address of the memory that needs to be fetched. The host processor initiates the execution by sending a start signal to the nZESPA controllers. Additional information like dimensions and number of channels of weight/activation matrices, initial addresses of the metadata (like activations, weights, strides, and bias) are offloaded to the register-stacks by the host processor. The bias register holds the bias value, which is added only once at each neuron position. The nZESPA modules can handle any stride value and thereby execute CNNs of diverse shapes and sizes.

TABLE I
CONFIGURATION PARAMETERS

nZESPA Parameters	Value	NMP-fully-sparse Parameters	Value
		# *nZESPA*s / grid	16
		# grids / vault	1
Multiplier width	16 bits	# grids in LoB	16
AB size	32 KB	# Multiplier	256
AIB size	8 KB	Total AB data	512 KB
WB size	256 B	Total AIB data	128 KB
WIB size	64 B	Total WB data	4 KB
RAB size	32 KB	Total WIB data	1 KB
RIB size	8 KB	Total RAB data	512 KB
		Total RIB data	128 KB

TABLE II
BENCHMARK DETAILS

ConvNets/Networks	AlexNet	VGG-16	ResNet-34
Year	2012	2014	2015
Place	1st	2nd	1st
Top-5 error	15.3%	7.5%	5.7%
# of CONV layer	5	13	33
# of parameters	61 M	138 M	21 M
Parameter Size	116.54 MB	263.90 MB	41.58 MB
Total Size	121.015 MB	373.58 MB	90 MB

TABLE III
SPECIFICATION OF THE ARCHITECTURES

Host Processor (common for all architecture)	
Core	x86-64
Core Count	4
Core Frequency	2 GHz
Cache Memory (common for all architecture)	
L1 i-cache	32 KB, private, 4 way associative, 64B blocks
L1 d-cache	32 KB, private, 8 way associative, 64B blocks
L2 cache	256 KB, shared, 8 way associative, 64B blocks
HMC (common for all architecture)	
Timings	t_{RP} = 7.7ns, tt_{CCD} = 3.3ns, t_{RCD} = 10.2ns, t_{CL} = 9.9ns, t_{WR} = 15ns, t_{RAS} = 21.6ns
Energy [18]	3.7 pJ/bit (DRAM read), 6.78 pJ/bit (SerDes hop)
Power [29]	11.08W
Size	2 GB
Logic Die	90nm, dimension $27 \times 27mm^2$
traditional-dense / traditional-fully-sparse	
# of grids	16 (for both architectures)
# of logic units (dense-accelerator/*nZESPA*) per grid	16 (for both architectures)
# of multipliers	256 (for both architectures)
Area overhead per grid	1.96 mm^2 / 2.56 mm^2
Frequency of dense-accelerator / *nZESPA* module	∼1.2 GHz / ∼1 GHz on UMC 90 nm
Capabilities on data	Parallel processing on dense / compressed-sparse
Area of integration	Near the on-chip memory controller, similar to [12]
NMP-dense / NMP-fully-sparse (proposed)	
# of grids	16 (for both architectures)
# of logic units (dense-accelerator/*nZESPA*) per grid	16 (for both architectures)
# of multipliers	256 (for both architectures)
Area overhead per grid	1.96 mm^2 / 2.56 mm^2
Frequency of dense-accelerator / *nZESPA* module	∼1.2 GHz / ∼1 GHz on UMC 90 nm
Capabilities on data	Parallel + NMP on dense / compressed-sparse
Area of integration	LoB of the HMC (for both architectures)

2) The hardware includes separate buffers for storing nonzero data vectors and index vectors of the compressed-sparse format, shown in Section III-C. The activation data and indices are stored in AB and activation index buffer (AIB), respectively. The weight buffer (WB) and weight index buffer (WIB) store the nonzero weights, and it's indices, respectively. The AB register (ABR) and WB register (WBR) are used to access the respective buffers since these registers hold the addresses for the corresponding buffer.
3) Each unit includes the 16-bit multiplier and adder to perform the MAC operations. The checker module plays a crucial role to achieve sparsity aware inference operations. It searches the AB efficiently for the valid input activations at each neuron position. As the AB only contains nonzero activations, the zero-valued activations are never fetched. It also finds the corresponding weights in the WB. Upon a successful search, the MAC operation is triggered. Note that WB also contains nonzero weights. Consequently, zero-valued weights are never fetched, and hence it possesses the ability to exploit both static and dynamic sparsity (fully sparse). The OAs are accumulated in the result accumulation buffer (RAB).
4) After the generation of the OA channels for all the filters, the post convolution module (PCM) applies the nonlinear activation function (ReLU) on the OA channels. On completion of ReLU, PCM performs the pooling operations and encodes the results in the specific compressed-sparse format for the processing of the next activation layer. PCM updates the RAB with only nonzero OA vectors and writes the corresponding index vector in the result index buffer (RIB). Finally, the OAs from RAB and OA indices (OAI) from RIB are updated in the DRAM die through the VC and TSVs.

Note that the multiple inputs of a MUX (e.g., MUX2-MUX6) are replaced by one thick arrow (blue lines) for the simplicity. Similarly, the multiple control signals between any two modules within the nZESPA are replaced by a single CTRL signal (green lines). The select lines of the multiplexers are shown in red lines. The configuration parameters of nZESPA and the NMP-fully sparse architecture are presented in Table I. The nZESPA design, mentioned above, preserves the nZESPA dataflow at every state of the state machine to leverage the sparsity close to the memory.

VI. EXPERIMENTAL METHODOLOGY

We evaluate all four architectures by executing widely used state-of-the-art benchmarks from industries and academia. The selected benchmarks stand in the top in terms of reduced *top-5 error rate* in the imagenet large-scale visual recognition challenge (ILSVRC) [28]. Table II lists the attributes of all the three *benchmarks*: 1) AlexNet [1]; 2) VGG-16 [19]; and 3) ResNet-34 [20]. The *top-5 error* of the networks has reduced substantially over time. The number of CONV layers has also increased, which makes the networks deeper and accurate. We have shown the parameter size and the total size of the model separately in Table II. The specification summary that includes shapes and sizes of the networks are extracted from the PyTorch framework [24].

A. Particulars of the Architectures

We compare four different architectures of a system: ❶ traditional-dense; ❷ traditional-fully sparse; ❸ NMP-dense; and ❹ NMP-fully sparse (proposed). The detailed specifications of these architectures are listed in Table III. All the architectures include a similar quad-core host processor with a similar cache hierarchy, as shown in Table III. HMC is used as the main memory module for all the architectures. The traditional systems have 16 grids of dense-accelerators, and 16 grids of nZESPA modules for the respective systems (❶ and ❷) integrated close to the on-chip main memory controller through a 2-D mesh NoC (similar place of integration as shown in [12]). Conversely, a grid of dense-accelerators

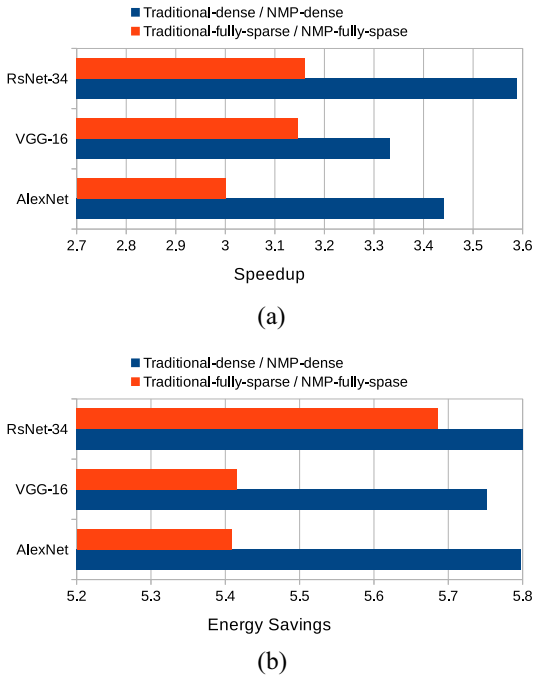


Fig. 9. Effectiveness toward NMP capability. (a) Speedup of dense and sparse systems due to NMP. (b) Energy savings on both dense and sparse systems due to NMP.

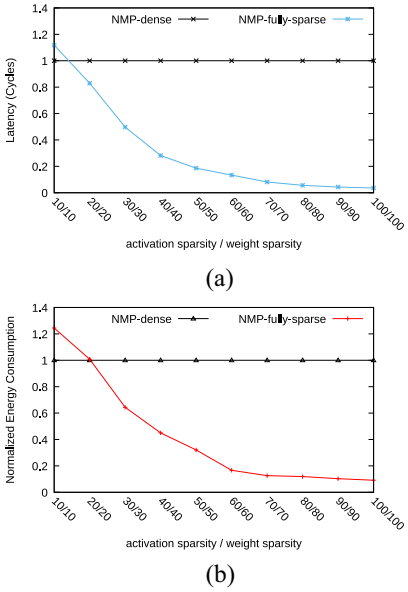


Fig. 10. ResNet-34 (a) performance and (b) energy versus sparsity.

is integrated with each vault of HMC in its LoB for NMP-dense architecture (❸). And the grid of nZESPA modules is integrated with each vault of HMC in the proposed NMP-fully sparse architecture (❹). Consequently, compared to the NMP-based architectures, all traditional architectures suffer from longer data access latency and higher per bit energy cost due to the high off-chip communication cost. The HMC parameters, capabilities, working frequency of dense-accelerators/nZESPA modules, and area overhead of the individual architecture are also mentioned in Table III. Note that, dense-accelerators are comparatively superior to the nZESPA modules in terms of area overhead and working frequency because sparse architectures require additional hardware and have a longer critical path for maintaining the sparse encoding. However, the property of sparsity neutralizes the effect, as shown in Section VI-C. The dense-accelerator follows a similar data partitioning scheme, like the data partitioning scheme of the proposed nZESPA dataflow. However, unlike nZESPA, it works on dense data, and no zero-valued computations are skipped. All these architectures are primarily designed to optimize the convolution layers because state-of-the-art CNNs are primarily dominated by these compute-intensive layers [21], [30]. However, these architectures include dedicated logic for the nonlinear operator (ReLU) and pooling. The detailed designs of the three baselines (traditional-dense, traditional-fully sparse, and NMP-dense) are not shown due to the limitation of space.

B. Measuring Performance and Power

1) Performance: We implement the designs of both the dense-accelerator and proposed nZESPA module in synthesizable Verilog Hardware Description Language (HDL). We then perform placement aware logic synthesis in Genus Synthesis Solution (version 15.21) from Cadence. We use UMC 90-nm technology library and obtain the operating frequency around 1.2 GHz and 1 GHz, respectively, for the dense-accelerator and proposed nZESPA module. We develop an in-house cycle level simulator which can be configured to resemble the state machine of the dense-accelerator and proposed nZESPA module. The simulator is parametrizable and can be fed with the frequency obtained from the hardware synthesis of the respective hardware. The execution time is measured from the simulator by layer-wise execution of the respective ConvNets. Note that the reported execution time by the simulator includes the data processing time in the grid of accelerators (dense/nZESPA) as well as the time to fetch/store results in the memory through the memory hierarchy of the respective architectures. Here, additional latency for off-chip communications in traditional systems is also taken into account. The timing parameters to fetch/store data in HMC are shown in Table III.

2) Power Analysis: The total power consumption of the dense-accelerator and nZESPA module is also obtained from the Genus Synthesis Solution (version 15.21) from Cadence after the logic synthesis. The power and energy parameters of the HMC device are obtained from the existing state-of-the-art (shown in Table III). We use McPAT [31] to measure the on-chip power values. While measuring the energy consumptions of each architecture during the execution of the ConvNets, we consider the energy consumed by the logic units (dense/nZESPA) as well as the energy required to fetch/store data through the memory hierarchy of the respective systems. Note that the area analysis for the dense-accelerator has been done in a similar fashion, as explained in Section VI-C5.

C. Evaluation

We have evaluated the proposed system (NMP-fully sparse) based on its performance, energy consumption, and area

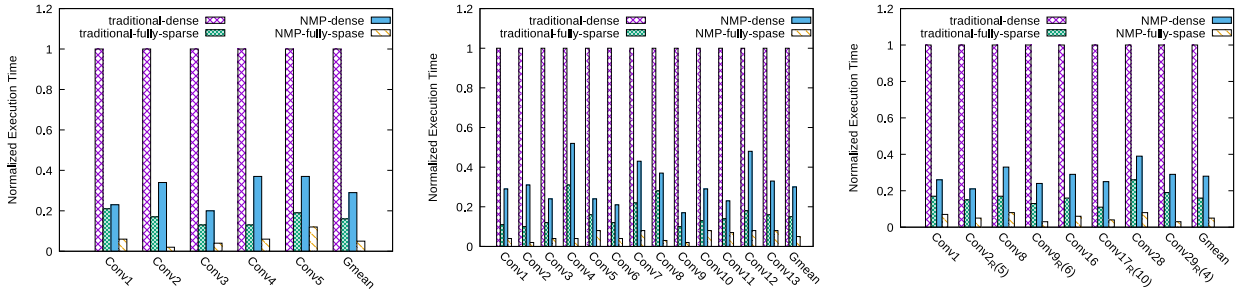


Fig. 11. Performance analysis of ConvNets/Networks; normalized w.r.t. traditional-dense architecture. (a) AlexNet CONV layers. (b) VGG-16 CONV layers. (c) ResNet-34 CONV layers.

overhead. The proposed system substantially outperforms all the baselines while being energy efficient. In this section, we first evaluate the effectiveness of NMP capability and sparsity for a system. Then we measure the system's throughputs for all the architectures.

1) Effectiveness of NMP Capability: Fig. 9 manifests the effectiveness of the NMP approach. The NMP capability benefits both the dense and sparse systems. In Fig. 9(a), we have shown the speedup achieved by the NMP-dense and NMP-fully sparse over the traditional-dense and traditional-fully sparse system, respectively, for all the networks. This speedup is obtained as the traditional systems (both dense and sparse) have longer off-chip memory access latency than the NMP-based systems. Note that, the speedup (traditional-dense/NMP-dense) achieved in the NMP-dense system due to NMP capability is higher than the speedup (traditional-fully sparse/NMP-fully sparse) obtained in the NMP-fully sparse system. The reason is; the number of data blocks that need to be accessed by the dense systems is higher than the sparse systems. Consequently, the amount of off-chip latencies, saved in the NMP-dense system, is higher than the NMP-fully sparse system. In Fig. 9(b), a similar effect can be seen in the energy savings due to the NMP capability. Both the NMP-dense and NMP-fully sparse have achieved significant savings in the energy compared to their respective traditional systems. The NMP-dense architecture leverages more benefits from the NMP approach than the NMP-fully sparse architecture due to its comparatively high data requirements.

2) Effectiveness of Sparsity: While executing the ResNet-34, we compare the performance and energy efficiency of the NMP-fully sparse with NMP-dense architecture while varying the activation and weight sparsity. We extract the data from PyTorch [24] and artificially wipe the weights and activations (similar to the process of [12]) in ResNet's layers to vary the sparsity in the range of 10%–100%. Along the *x*-axis of Fig. 10, we plot activation/weight sparsity, which scales simultaneously. For example, the point 50/50 represents 50% weight and 50% activation sparsity. Fig. 10 shows that at 10% (10/10 point) sparsity, NMP-fully sparse achieves about 89% of the performance of NMP-dense architecture because of the complexity of maintaining the sparse encoding of the nZESPA dataflow. However, NMP-fully sparse starts performing better than the NMP-dense with an increasing amount of sparsity. At point 20/20, NMP-fully sparse obtains 1.2× better performance than NMP-dense, which reaches up to 27× at point 100/100. A similar trend is also observed in the case of energy consumption in Fig. 10(b). At point 10/10, NMP-fully sparse achieves only 80% of the energy efficiency of NMP-dense architecture due to the additional hardware overhead to maintain sparse encoding. With the increasing amount of sparsity, the NMP-fully sparse starts obtaining better energy efficiency, which reaches up to 10× at point 100/100. Certainly, increasing the sparsity in weights and activations affects the classification accuracy. For ResNet-34 at point 10/10, we observe that the accuracy is very close to the accuracy of the original dense network. Here, the loss of accuracy is minimal (below 1%). At point 50/50, this loss reaches only up to 10%. However, it can reach to complete loss of accuracy at point 100/100. Note that point 100/100 is a hypothetical situation and does not occur typically in the piratical scenario.

TABLE IV
AVERAGE SPEEDUP OVER RESPECTIVE BASELINES

Networks	Speedup over baselines		
	traditional-dense	traditional-fully-sparse	NMP-dense
AlexNet	18.48x	3x	5.37x
VGG-16	20.30x	3.14x	6.09x
ResNet-34	19.44x	3.16x	5.41x

3) Performance Analysis: We compare the performance of the proposed NMP-fully sparse architecture with all three baselines discussed in Section VI-A. In Fig. 11, the normalized execution times with respect to (w.r.t.) traditional-dense architecture are plotted along the *y*-axis while all the CONV layers of the respective networks are shown on the *x*-axis. Separate layer-wise performance analysis for AlexNet, VGG-16, and ResNet-34 are presented in 11(a)–(c), respectively. The proposed NMP-fully sparse system outperforms all the baselines in terms of execution time. To measure the performance, we consider both the execution time as well as data fetching/storing time through the memory hierarchy of the respective systems. Table IV shows the average speedup achieved by the proposed system over the respective baselines. We have achieved the maximum speedup over traditional-dense architecture (see the first column of Table IV). The reasons are: 1) traditional-dense architecture works on dense data. Consequently, no zero-valued computations are skipped here

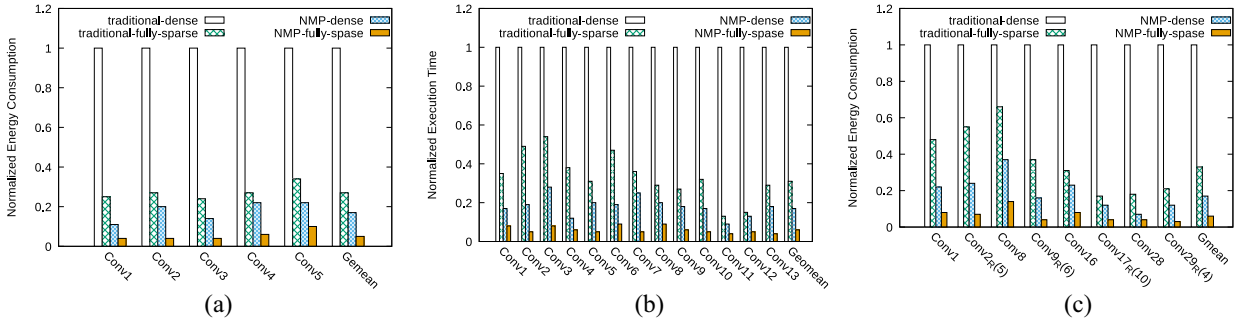


Fig. 12. Energy efficiency comparison of ConvNets/Networks; normalized w.r.t. traditional-dense architecture. (a) AlexNet CONV layers. (b) VGG-16 CONV layers. (c) ResNet-34 CONV layers.

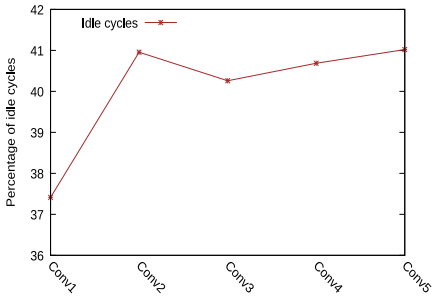


Fig. 13. Percentage of Idle cycles for multipliers (AlexNet CONV layers).

and 2) it does not own the NMP capability and hence suffers from high off-chip latency. The gain over the traditional-fully sparse architecture (the second column of Table IV) is comparatively less as it owns the similar property of exploiting sparsity for both the weights and activations. The gain over this architecture stems out only from the additional NMP capability of the proposed system. Compared to traditional-fully sparse, the proposed system provides relatively higher speedup in the case of NMP-dense (the third column of Table IV) as it is dense, and no computations are eliminated in NMP-dense architecture. However, this NMP-dense architecture performs better than the traditional-dense since it possesses the NMP capability like the proposed system. Note that, ResNet-34, shown in Fig. 11(c), has 33 CONV layers. We have only shown the representative CONV layers. For example, the term, $\text{Conv2}_R(5)$, is the representative of the subsequent five layers.

The idle cycles for the multipliers is an overhead for the sparse accelerators. Fig. 13 quantitatively shows the percentage of idle cycles for the CONV layers of the AlexNet. The y-axis of Fig. 13 shows the percentage of idle cycles for the multipliers of the nZESPA modules, while the x-axis corresponds to the CONV layers of the AlexNet. The idle cycles for the multipliers are primarily consumed by the checker modules and the controllers of the nZESPA modules before it sends the weight and activation pairs to the multipliers. Compared to the dense accelerators, this overhead is higher in the case of sparse accelerators because of the complexity of maintaining the sparse encoding.

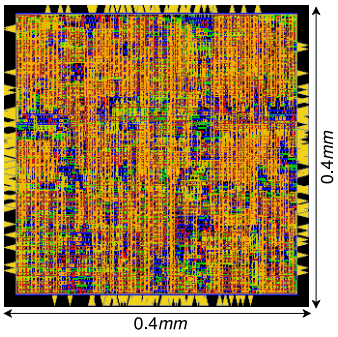
4) Energy Efficiency: Fig. 12 presents the comparison of the four accelerated architectures across the layers of the ConvNets. In Fig. 12, the normalized energy consumptions

TABLE V
AVERAGE SAVINGS IN ENERGY OVER RESPECTIVE BASELINES

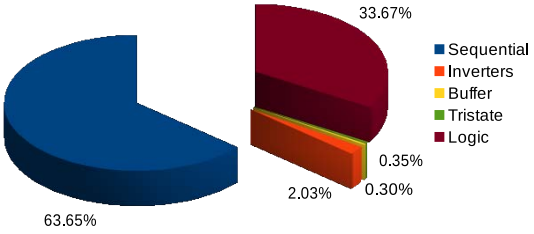
Networks	Energy efficiency over baselines		
	traditional-dense	traditional-fully-sparse	NMP-dense
AlexNet	19.9x	5.4x	3.44x
VGG-16	17.35x	5.41x	3.01x
ResNet-34	17.49x	5.69x	3.02x

w.r.t traditional-dense architecture are plotted along the y-axis, and the CONV layers of the individual networks are shown on the x-axis. The proposed NMP-fully sparse architecture harnesses substantial energy efficiency over all three baselines. We consider the total energy consumption of each system. The total energy consumption includes the energy consumption of each unit as well as the energy required to fetch/store data through the memory hierarchy of the respective systems. Table V illustrates the average improvement in energy consumption achieved by the proposed system over the respective baselines. The maximum saving in energy is obtained over traditional-dense (shown in the first column of Table V) architecture as it neither exploits the sparsity nor the NMP approach. The improvement in the energy efficiency over traditional-fully sparse architecture is achieved because of the lack of NMP capability in it, while the energy efficiency over NMP-dense architecture is obtained as it does not exploit the sparsity (shown in the second and third column of Table V). Note that, similar to Fig. 11(c), we show only the representative CONV layers in Fig. 12(c).

5) Area Analysis: Memory industries are area-sensitive. Consequently, area overhead plays a crucial role in in-memory integration of the logic. We implement the nZESPA hardware in Verilog HDL. We use the Innovus (from Cadence toolset) to obtain post-synthesis area estimates. Fig. 14(a) depicts the layout of the designed nZESPA, where the standard cells are placed. Our nZESPA net-list gets placed on $0.4\ \text{mm} \times 0.4\ \text{mm}$ [shown in Fig. 14(a)] square box at UMC 9-nm technology. This leads to an area overhead of 2.56 mm^2 for a single grid, resulting in a 5.61% of overhead for all 16 grids in total. Fig. 14(b) shows the area breakups of a single nZESPA module. Note that the sequential elements, including AB, AIB, WB, WIB, RAB, and RIB of the nZESPA module (shown in Fig. 8), consume the maximum area which is around 63.65%. The second highest consumer of the area is the logic elements, which incur an overhead of approximately 33.67%.



(a)



(b)

Fig. 14. Area analysis. (a) nZESPA-chip layout. (b) Area breakdown of the designed nZESPA.

6) Thermal Feasibility: The designed nZESPA modules and its specification provide us with ease from the thermal concerns. In the previous studies [26], it has been investigated that an HMC with 4-stacked DRAM die with a processing cluster clocked up to 5 GHz can be integrated into the logic die. This integration fits within the thermal limit of HMC [32] as this integration increases the temperature up to 76 °C. Apart from that, thermal feasibility in the case of near-memory integration of an 8.5 W processors, is also manifested in [33]. The total power budget of nZESPA modules is 2.85 W, which is less than the power budget (3.41 W) of [26] in a similar technology node. This budget is also substantially less than that of 8.5 W. Similar to [14] and [34], we can also safely conclude that our proposed architecture is thermally feasible as it has a lower power budget and frequency.

D. Comparison With Previous Accelerators

In this section, we compare the proposed NMP-fully sparse architecture with the architectures that exploit the static (NMP-static-sparse) and dynamic sparsity (NMP-dynamic-sparse) in isolation. The existing state-of-the-art architectures like Cnvlutin (dynamic sparsity) [9] and Cambricon-X (static sparsity) [10] have leveraged the benefits of exploiting static and dynamic sparsity in isolation. Analogous to the principles of Cambricon-X [10], we implement the NMP-static-sparse architecture, which utilizes the static sparsity property during the execution of CNN tasks near the HMC. Analogous to the principles of Cnvlutin [9], we also design NMP-dynamic-sparse that exploits the dynamic sparsity property during the execution of CNNs in the LoB of the HMC. Due to the substantial differences in the organization/buffering size, design and dataflow, and design choices like the use of eDRAM, the implemented architecture can not precisely replicate the state-of-the-art. However, we keep the spirit of NMP-static-sparse and NMP-dynamic-sparse the same as the Cambricon-X and Cnvlutin, respectively. A similar approach of comparison has also been performed in SCNN [12].

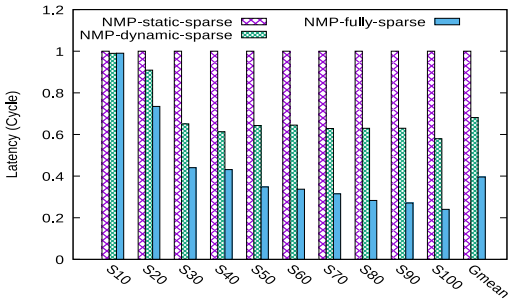


Fig. 15. Performance comparison.

Fig. 15 shows the comparison of the performance while executing ResNet-34 in the three different architectures. We plot the varying sparsity along the x-axis. For example, the point S10 represents 10% static sparsity in NMP-static-sparse, 10% dynamic sparsity in NMP-dynamic-sparse, and 10% sparsity in both the weights and activations for the proposed NMP-fully sparse architecture. The proposed system performs well in all the points. On average, the NMP-fully sparse achieves 2.5× and 1.7× better performance than the NMP-static-sparse and NMP-dynamic-sparse architecture, respectively. Note that this gain is initially less due to the minimum amount of sparsity, and the gain increases for the higher sparsity values. Analogous to the performance comparison, we observe a similar trend in the comparison of energy consumption by these systems. Note that our proposal will perform even better when static-sparse and dynamic-sparse systems are integrated into traditional (on-chip) setting instead of near-memory.

Interestingly, the benefits of proposed NMP-fully sparse architecture can also be seen in the following observation. There are differences between the architecture of SCNN [12] and the proposed one.

1) Though SCNN targets both static and dynamic sparsity like us, they implement on-chip accelerators while the proposed one is a near-memory solution.
2) SCNN uses a Cartesian product-based solution with an assumption of unit strides for convolution, when unit stride may not always be true for all the hidden layers of different ConvNets. This Cartesian-product strategy restricts SCNNs applicability only to CNNs with unit-stride convolutions. However, the proposed architecture relies on the dot-products and can be used to accelerate any CNNs with any strides.
3) SCNN keeps the input_activations in the PEs while it broadcasts the filter elements to all the PEs.

As different input_activations inevitably have different sparsities, and because all input maps are multiplied by the same filter, the PEs with denser maps would lag behind those with sparser maps by the next broadcast of the filter. This approach increases the systematic load imbalance while providing filter reuse benefits. Unlike the filter broadcast,

the proposed nZESPA dataflow keeps the filter in the local buffers (WB, WIB of Fig. 8) of the nZESPA modules and consequently survives from such a load imbalance issue in exchange for a loss of filter reuse. It can be observed that the proposed NMP-fully sparse architecture achieves several times more performance gain over the traditional-dense architecture compared to SCNN. While SCNN delivers 2.37× and 3.52× performance again over the traditional-dense system for AlexNet and VGGNet, respectively, the proposed NMP-fully sparse architecture obtains 18.48× and 20.30× performance gain over the traditional-dense system for the same networks. This additional gain is achieved due to the minimum memory access latency of NMP as well as for the efficiency of nZESPA dataflow.

VII. Related Work

Several research projects like Diannao family [5], [35], [36], Eyeriss [6], TPU [7], NeuFlow [37], SmartShuttle [38], FlexFlow [39], and so on, have harnessed the parallelism and thereby accelerated the CNN operations. Eyeriss [8] exploits both the parallelism and sparsity while executing the CNNs. However, only the dynamic sparsity has been considered here. The multipliers are gated on detecting the zero-values in input activations. This gating logic provides savings in energy consumptions, but it does not improve the performance. Eyeriss uses run-length coding (RLC) for data compression, which improves the system's overall throughput (performance + energy consumption). Our NMP-fully sparse uses a variant of a compression scheme that is used in [13] and [25]. Unlike Eyeriss, we maintain the compressed format of data for the entire inference processing in the internal buffers of nZESPA. In addition to this, we explore sparsity in both the weights and activations. Cambricon-X [10] and GPU ZeroSkip [11] have only investigated the static sparsity in their work. Cnvlutin [9] is another work where the dynamic sparsity has been exploited. In [40], only sparsity of the operands has been used to expedite its performance. The zero-valued compression technique is not incorporated here. The EIE [13] and SCNN [12] are the ones that target sparsity maximally while eliminating the computations for both zero-valued weights and activations. However, EIE targets the fully connected layer of the CNN algorithm. Apart from that, though EIE targets sparsity in both the activations and weights for fully connected layers, the performance of EIE is equivalent to the one-sided sparsity only. The reason is: it discards zeros in the weights but incurs compute-idling due to the discard. SCNN is more aggressive in a sense as it targets the optimization of the convolution layers, which comprise the maximum number of computations. In addition to this, SCNN also employs the compression technique for both the weights and activations. The compressed form of the data is maintained everywhere in the design until the end of all computations. Despite all of these, SCNN still involves costly off-chip DRAM accesses while loading activations to its internal buffer (IARAM) and weights to the WB. Unlike SCNN, the proposed architecture (NMP-fully sparse), having NMP capability, avoids these off-chip communications as all the data are processed in the logic layer of the HMC device. Some other projects like Neurostream [14], PIM [41], CMP-PIM [15] have incorporated the concept of NMP while executing the DNNs/CNNs for dense systems.

VIII. Conclusion

In this article, we proposed hardware design of nZESPA and integrate it into a versatile architecture, NMP-fully sparse, which possesses capabilities like exploiting parallelism, utilizing the sparsity of both weights as well as activations, and near-3D-memory processing. The cumulative outcome from all these features makes this architecture aggressive to outperform all other baselines while executing state-of-the-art benchmarks like AlexNet, VGG-16, and ResNet-34. On average, the proposed system obtains maximum up to 20.30×, 3.16×, and 6.09× improvement in the performance over the traditional-dense, traditional-fully sparse, and NMP-dense architecture, respectively. Additionally, on an average, the proposed system achieves maximum up to 19.9×, 5.69×, and 3.44× energy efficiency, compared to the traditional-dense, traditional-fully sparse, and NMP-dense architecture, respectively. On UMC 90-nm technology, the combined area overhead for all the 16 grids of nZESPA modules is only 5.61%. Certainly, the obtained results make the proposed NMP-fully sparse architecture an attractive and deserving candidate for practical implementation.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2012, pp. 1097–1105.

[2] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with COTS HPC systems," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.

[3] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, p. 18.

[4] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 646–661.

[5] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, Cambridge, U.K., 2014, pp. 609–622.

[6] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, 2016.

[7] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 1–12.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[9] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 1–13, 2016.

[10] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, Taipei, Taiwan, 2016, p. 20.

[11] H. Park, D. Kim, J. Ahn, and S. Yoo, "Zero and data reuse-aware fast convolution for deep neural networks on GPU," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ ISSS)*, 2016, pp. 1–10.

[12] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2. pp. 27–40, 2017.

[13] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.
[14] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 420–434, Feb. 2018.
[15] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "CMP-PIM: An energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proc. 55th ACM/ESDA/IEEE Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2018, p. 105.
[16] P. Das, S. Lakhotia, P. Shetty, and H. K. Kapoor, "Towards near data processing of convolutional neural networks," in *Proc. 31st Int. Conf. VLSI Design 17th Int. Conf. Embedded Syst. (VLSID)*, Pune, India, 2018, pp. 380–385.
[17] S. Sen, S. Jain, S. Venkataramani, and A. Raghunathan, "SparCE: Sparsity aware general-purpose core extensions to accelerate deep neural networks," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 912–925, Jun. 2019.
[18] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Proc. IEEE Symp. VLSI Technol. (VLSIT)*, Honolulu, HI, USA, 2012, pp. 87–88.
[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.
[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.
[21] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 3431–3440.
[22] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.
[23] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*. Montreal, QC, Canada: Morgan Kaufmann Publ., Inc., 2015, pp. 1135–1143.
[24] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *Proc. 31st Conf. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, 2017, pp. 1–4.
[25] R. W. Vuduc and J. W. Demmel, "Automatic performance tuning of sparse matrix kernels," Dept. Comput. Sci., Univ. California, Berkeley, CA, USA, 2003.
[26] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *Proc. ACM/IEEE 43rd ACM/IEEE 43rd Annu. Comput. Archit.ISCA)*, Seoul, South Korea, 2016, pp. 380–392.
[27] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," *ACM SIGOPS Oper. Syst. Rev.*, vol. 51, no. 2, pp. 751–764, 2017.
[28] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[29] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proc. IEEE Hot Chips 23 Symp. (HCS)*, Stanford, CA, USA, 2011, pp. 1–24.
[30] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014. [Online]. Available: arXiv:1404.5997.
[31] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchit.*, New York, NY, USA, 2009, pp. 469–480.
[32] "HMC specification 2.1, hybrid memory cube consortium," Micron Technol., Boise, ID, USA, Rep., 2015.
[33] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," in *Proc. Workshop Near Data Process. (WoNDP)*, 2014, pp. 1–5.
[34] A. Pattnaik *et al.*, "Scheduling techniques for GPU architectures with processing-in-memory capabilities," in *Proc. Int. Conf. Parallel Archit. Compilation*, 2016, pp. 31–44.
[35] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 269–284, 2014.
[36] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.
[37] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, Colorado Springs, CO, USA, 2011, pp. 109–116.
[38] J. Li *et al.*, "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," in *Proc. IEEE Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Dresden, Germany, 2018, pp. 343–348.
[39] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, 2017, pp. 553–564.
[40] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, New Orleans, LA, USA, 2017, pp. 2861–2865.
[41] L. Xu, D. P. Zhang, and N. Jayasena, "Scaling deep learning on multiple in-memory processors," in *Proc. 3rd Workshop Near Data Process.*, 2015, pp. 1–7.

Palash Das (Member, IEEE) received the B.Tech. degree in computer science and engineering (CSE) from the Dumkal Institute of Engineering and Technology, Basantapur, India, in 2009, and the M.E. degree in CSE from the Indian Institute of Engineering Science and Technology Shibpur, Shibpur, India, in 2013. He is currently pursuing the Ph.D. degree in CSE with the Indian Institute of Technology Guwahati, Guwahati, India.

His research interest includes near-data processing.

Hemangee K. Kapoor (Senior Member, IEEE) received the B.Eng. degree in computer engineering from the College of Engineering Pune, Pune, India, in 1998, the M.Tech. degree in computer science and engineering (CSE) from the Indian Institute of Technology (IIT) Bombay, Mumbai, India, in 2000, and the Ph.D. degree in CSE from London South Bank University, London, U.K., in 2004.

She is currently a Professor with the Department of CSE, IIT Guwahati, Guwahati, India. Her research interest includes multiprocessor computer architecture.

Prof. Kapoor is a Senior Member of ACM.