Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory

Duckhwan Kim*, Jaeha Kung*, Sek Chai†, Sudhakar Yalamanchili *, and Saibal Mukhopadhyay*
*School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA
kimduckhwan, jhkung, sudha, smukhopadhyay6@gatech.edu
†SRI International, Princeton, New Jersey, USA
sek.chai@sri.com

***Abstract*—This paper presents a programmable and scalable digital neuromorphic architecture based on 3D high-density memory integrated with logic tier for efficient neural computing. The proposed architecture consists of clusters of processing engines, connected by 2D mesh network as a processing tier, which is integrated in 3D with multiple tiers of DRAM. The PE clusters access multiple memory channels (vaults) in parallel. The operating principle, referred to as the memory centric computing, embeds specialized state-machines within the vault controllers of HMC to drive data into the PE clusters. The paper presents the basic architecture of the Neurocube and an analysis of the logic tier synthesized in 28nm and 15nm process technologies. The performance of the Neurocube is evaluated and illustrated through the mapping of a Convolutional Neural Network and estimating the subsequent power and performance for both training and inference.**

***Keywords*-Neural nets; Neurocomputers; Neuromorphic computing**

I. Introduction

Computational efficiency is a primary concern in neuro-inspired learning systems. While there are important advances in Deep neural networks enabling new capabilities in visual search, signal processing, data mining, and other big-data applications [1], these systems are still very complex and it can take many weeks to train neural nets with today's computing systems. General Purpose Graphics Processing Units (GPGPU) have driven much of the recent success in Deep Learning, enabling key algorithm explorations with sufficient level of acceleration [2]. Researchers are also exploring other computing fabrics such as FPGAs [3]–[5], and more recently, neuromorphic hardware ASIC accelerators [3]–[8] to improve the computational performance and power efficiency of neuro-inspired algorithms. The programmability of GPGPUs supports realizations of different types and scale of neural nets but have much lower power-efficiency. On the other hand, ASICs have much higher power and computational efficiency, but are not programmable and consequently not scalable. The goal of this paper is to propose and evaluate an architecture that can provide GPGPU like programmability/scalability but with higher power and computational efficiency (closer to that of ASICs).

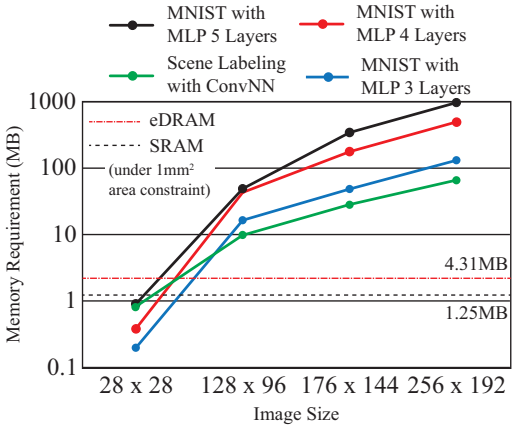

Figure 1. Required memory for scene labeling [9] with different image size using convolutional neural network and for MNIST [10]. Memory capacity of SRAM and eDRAM is normalized by $1mm^2$ area constraint [11], [12].

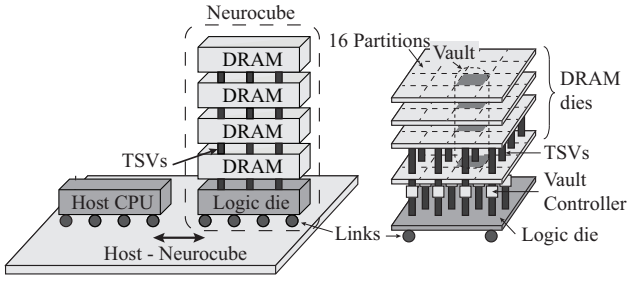

Figure 2. Neurocube architecture using Micron's Hybrid Memory Cube.

Neural network is highly parallel, with opportunities to exploit data and thread level parallelism. While there are architectures that attempt to achieve performance and scalability by reducing the processing units [3]–[5], [7], [8], the overall system performance is often limited due to insufficient memory bandwidth and network latency. Low operational density (ops/byte), poor spatial locality, and massive parallelism serve to stress memory bandwidth.

As a result, early on-chip cache solutions [4], [8] were tried but they were not scalable to large neural networks

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.41

or complex applications over large data sets, e.g., high resolution image and extracted features [4]. For example, Fig. 1 shows the growth in required memory capacity for scene labeling [9] with different input image sizes using convolutional neural network [10] and MNIST [10] with MLP [13]. It shows even the use of high density eDRAM on-chip cache memory [7] cannot support large input image sizes and/or deeper leaning networks on-chip underscoring the pressure on off-chip memory bandwidth.

This paper introduces the Neurocube: a programmable, scalable, power efficient digital architecture platform for computing neuro-inspired algorithms (Fig. 2). Our approach is based on following key innovations:

1. In-memory neuromorphic processing. The Neurocube integrates a fine grained, highly parallel, compute layer within a 3D high-density memory package, the hybrid memory cube (HMC).

2. Memory-centric neural computing (MCNC). We exploit the data-driven nature and statically known memory access patterns of neuro-inspired algorithms to implement a programmable memory system to drive data flow enabled compute units. The lack of a traditional instruction set model is a major source of energy and compute efficiency improvement.

3. Programmable Neurosequence Generator. Host programming interface is a set of memory-based programmable state machines (neurosequence generators) that exposes abstractions for connectivity and synaptic weights as the vehicle for programming different neural nets.

We analyze performance and power of Neurocube for both training and inference of ConvNN (Scene Labeling [9]). Neurocube synthesized with 15nm can deliver throughput up to 132GOPs/s during inference and 127GOPs/s during training within a compute power envelope of 3.4W in 15nm FinFet. The simulations project ∼4X improvement in computing power-efficiency (GOPs/s/W) over reported GPU based implementation while providing the programmability and scalability advantages over ASIC/FPGA platforms.

The remainder of the paper is organized as follows: Section 2 introduces preliminaries for Neurocube; Section 3 describes the architecture of Neurocube; Section 4 introduces memory centric neural computing; Section 5 describes the operation of Neurocube; Section 6 simulates system performance; the hardware is synthesized with 28nm and 15nm process in Section 7; Section 8 compares Neurocube with previous work, and concludes in Section 9.

II. Preliminaries

A. Programmable Digital Neural Network

In this paper, we will use the term *neural network* (NN) to represent an artificial neural network [22]. In general, NN is composed of multiple layers of neurons where each layer is composed of multiple neurons. The first layer, which receives the raw input (e.g., image), is called the input layer.

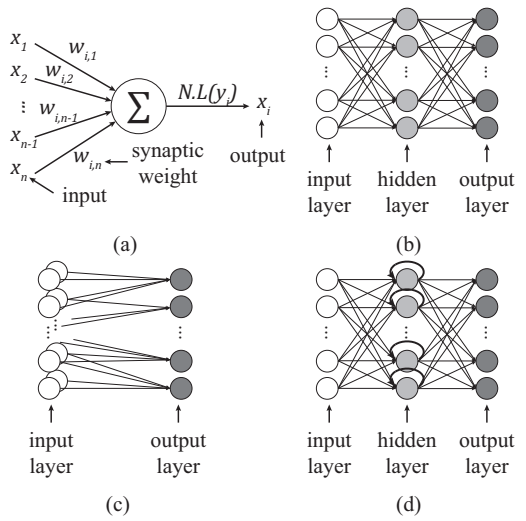

Figure 3. (a) Simple neuron diagram, (b) Fully-connected feedforward composed of input layer, one hidden layer, and output layer, (c) Convolutional neural network (feedforward, sparse connection), and (d) Recurrent neural network (fully-connected feedback).

The last layer, which generates the output of the NNs, is called the output layer. The multiple layers between the input and the output layers are referred to as the hidden layers.

Fig. 3 (a) illustrates the neuron, which state (y_i) is defined as summation of weighted (w_{ik}) inputs (x_k) and the threshold is represented as an activate function ($N.L(y)$). The state (y_i) and output (x_i) of neuron i is computed as

$$y_i = \sum_k w_{ik} \cdot x_k \tag{1}$$

$$x_i = N.L(y_i). \tag{2}$$

where, $N.L()$ is a non-linear activate function. The basic operation of a single neuron is common to all NNs (Eq. 1) while the activate function may differ. Different NNs can be defined by the connections of the neurons in the network. For example, a multi-layer perceptron (MLP [13]) is a feedforward network in which each neuron in one layer is connected to all neurons in the next layer (Fig. 3 (b)). In Eq. 1, k refers to all connected neurons in the preceding layer. For a convolutional neural network (ConvNN [10]), k is the 2D-neighborhood of i; therefore only a few neurons placed locally together are connected to a neuron in next layer (Fig. 3 (c)). For a recurrent neural network (RNN [23]), k refers to all neurons in the preceding layer and includes itself (recurrent); the current output is the input to compute the state at the next time step (Fig. 3 (d)).

We should note that the main difference between different neural networks is the *set k* - the set of neurons connected to a single neuron in the next layer. In other words, each class of neural networks can be defined by the connectivity between neurons, while the basic operation of a neuron remains

Table I
3D STACKED MEMORY SPECIFICATION.

	DDR3 [14]	Wide I/O 2 [15]	HBM [16]	HMC-Ext [17]	HMC-Int [17]
Interface	2D	3D	2.5D	3D	3D
Max. # Channels	2	8	8	8	16
Word Size	64 bit	128 bit	128 bit	32 bit	32 bit
Peak B.W.†	12.8 GBps	6.4 GBps	16 GBps	40 GBps	10 GBps
t_{CL} + t_{RCD}	25 ns	N/A	N/A	27.5 ns [18]	27.5 ns [18]
Operating Voltage	1.5 V, 1.35 V [14]	1.1 V [15]	1.2 V [19]	1.2 V [20]	1.2 V [20]
Energy	70 pJ/bit [21]	N/A	N/A	10 pJ/bit [20]	3.7 pJ/bit [20]

†*Peak bandwidth per channel*

the same - (*multiplication and accumulation*). Therefore, we observe that a group of multiply-accumulate (MAC) units can be used to emulate a range of neural network sizes and types by re-programming the connectivity between the units. We further observe that programmable connectivity simply means orchestrating the required data flows between memory and the MAC units.

B. 3D High-Density Memory

Fig. 1 underscores the need for both memory size and bandwidth. High density 3D memory, composed of multiple stacked DRAM dies offers to meet the capacity and bandwidth demands of neuro-inspired computations. Table I compares several candidate 3D memory technologies.

Wide I/O-2 is designed for mobile platforms by stacking conventional DRAM on the mobile SoC (3D interface) [15]. Using high density TSVs, the number of I/Os per channel is high (total number of I/Os from 8 channels is 1,024). High Bandwidth Memory (HBM) is designed for high performance processors [19]. HBM is composed of 4 DRAM dies and one single logic die. The logic die is designed for testing (design for test (DFT)), TSV arrays, and interface (PHY) for communication with the SoC.

The Hybrid Memory Cube (HMC) is also designed for high performance applications [17]. It is composed of multiple stacked DRAM dies and a single base logic die interconnected with TSVs. Each DRAM die is divided into 16 partitions in a 2D grid and the corresponding partitions on the vertical die form a single *vault*. Each vault has an independent vault controller on the logic die; therefore multiple partitions in the DRAM die can be accessed simultaneously. There have been proposals for off-loading data-intensive operations onto the logic die [24]–[26].

Compared to HBM or Wide I/O-2, the HMC architecture provides highly parallel access to the memory (one channel per vault) which is better suited to the highly parallel architecture of the computing layer in the Neurocube. The logic and memory dies can be fabricated in different process technologies; e.g., DRAM dies are fabricated in 50nm and logic die is fabricated in 28nm [20]. However, the area of the logic die relative to the memory dies is constrained by the package [20], and power dissipation is limited by the much tighter thermal constraints [27]. In this paper, we will refer to the interface between DRAM layers and the logic die as 'HMC-Int' (6^{th} column in Table I).

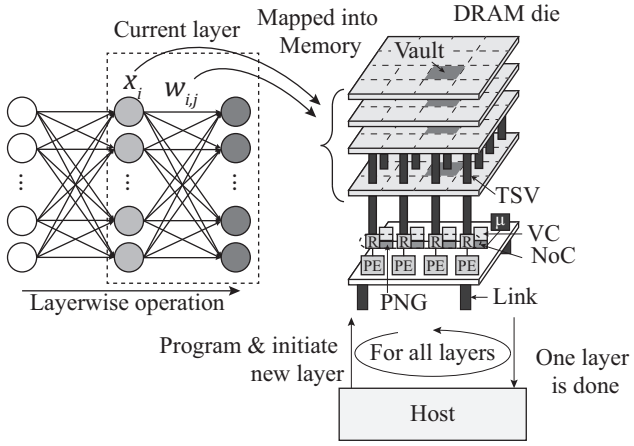

Figure 4. Programming Neurocube flow.

C. Programming and Execution Model

Fig. 4 illustrates a high level model of the programming and execution of a NN using the Neurocube architecture. The NN shown in Fig. 4 is stored in the DRAM stack - the locations of i) the layers, ii) the states of all of the neurons, iii) the corresponding connection weights are known a priori. Consequently, the data movement paths required between the DRAM layers and the logic layer to operate the network are known a priori. These data movement paths are compiled into state machine descriptions that will drive programmable neurosequence generators (PNG) integrated with the vault controllers. The host initiates the processing of each layer by programming the PNG by loading the state machines which streams data to the compute layer which operates in a data-driven manner. Thus, the execution of each NN layer is fully data driven. We refer to this as a *memory-centric neural computing* model since the compute units are driven by the memory system and there is no stored program in the traditional sense. The remainder of the paper describes each of the microarchitectural elements in greater detail.

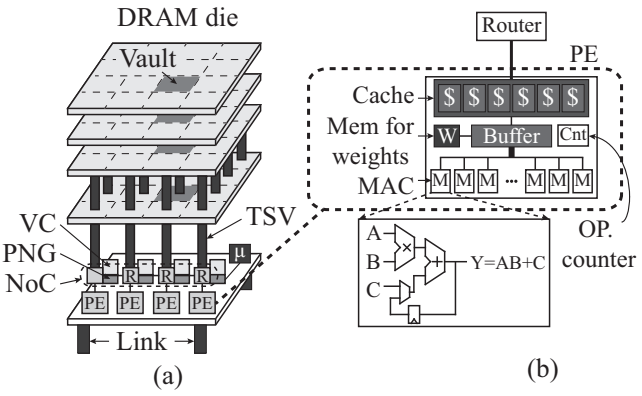

Figure 5. (a) Proposed Neurocube architecture and (b) Organization of the processing elements (PEs).

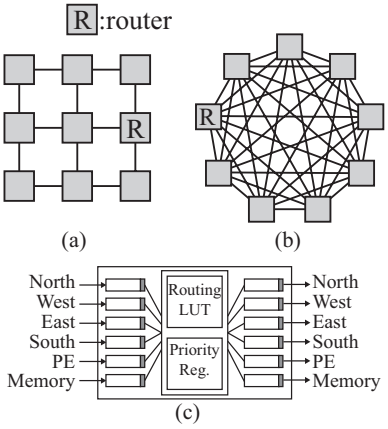

Figure 6. (a) 2D mesh NoC, (b) 2D fully connected NoC, and (c) Router design for 2D mesh NoC.

III. Neurocube Architecture

Fig. 5 illustrates the key components of the Neurocube architecture designed in the logic die of a HMC. Multiple processing elements (PE) concurrently communicate with multiple DRAM vaults through high-speed TSVs. A host communicates with the Neurocube through external links of the HMC to configure (program) the Neurocube for different neural network architectures (such as number of layers, types of layers, and dimension of layers). The Neurocube architecture is composed of a global controller, programmable neurosequence generator (PNG) for DRAM, routers for a 2D-mesh network on chip, and processing elements (PEs). Notations to describe the architecture are explained as: Number of DRAM banks (vaults or channels): n_{Ch}, Number of PE per DRAM bank (vault): n_{PE}, Number of MACs per PE: n_{MAC}, DRAM I/O clock frequency: $f_{DRAM-IO}$, NoC router clock frequency: f_{NoC}, PE clock frequency: f_{PE}, MAC clock frequency: f_{MAC}.

A. Memory System

Fig. 2 shows the structure of HMC described in Section II-B. To utilize all 16 vaults effectively, each vault is connected to one PE. Note that in general we can have a different number of MAC units/PE to match the vault bandwidth. All PEs are connected by a 2D mesh network.

B. Processing Elements (PE)

The processing element (PE) is the main computing unit and is comprised of multiple multiply accumulator (MAC) units since weighted summation is the main arithmetic operation in the emulated NNs (Eq. 1). A single PE is composed of n_{MAC} MAC units, a cache memory, a temporal buffer, and a memory module for storing shared synaptic weights. Fig. 5 (b) shows a block diagram of a single PE.

1) Multiply-Accumulator: In this paper, we used a 16-bit fixed point ($Q_{1,7,8}$: 1bit MSB, 7bits integer, 8bits fractional part) representation for both the state and weights in a generic neural network. The operating clock frequency of a MAC (f_{MAC}) is determined as below

$$f_{MAC} = f_{PE}/n_{MAC} \tag{3}$$

where f_{PE} = f_{NoC} = $f_{DRAM-IO}$ (operating frequencies of the PE, NoC, and DRAM I/O system). To compute sum of multiplications ($\sum W \times X$), the output of a MAC needs to be used as an input in next cycle. (Fig. 5 (b)).

2) PE Memory: The operation of the PE and the role of the various memories are best illustrated with an example. Consider a network where each layer has 8 neurons and each neuron has 3 input neurons from the previous layer. The PE has 8 MAC units. Consider the update of a layer. The 8 MAC units in a PE synchronously process/update one output neuron at a time. On cycle 1, each MAC unit computes the summation with the first input of each neuron. On cycle 2, each MAC unit computes the second term of the summation using the state and weight from its second input neuron and so on. Thus in three cycles the states of all 8 neurons have been updated.

The state of the input neurons and their associated connectivity weights are encapsulated in a packet and moved to the PEs by the programmable PNGs. Each packet has an OP-ID value to indicate whether they are the first, second, etc. input for its corresponding output neuron. Each PE has an operation counter (OP-counter) used to sequence through the correct number of input neurons for each output neuron whose state is being updated. At any point in time, the OP-counter refers to the input number currently being computed by the MAC units (i.e., input 1, input 2 etc.). If packets arrive out of order they are buffered in the SRAM cache until all corresponding inputs arrive, i.e., it is buffered if the OP-ID of the packet is greater than the OP-counter value. When all corresponding inputs arrive, they are moved to the temporal buffer and the availability of all inputs triggers

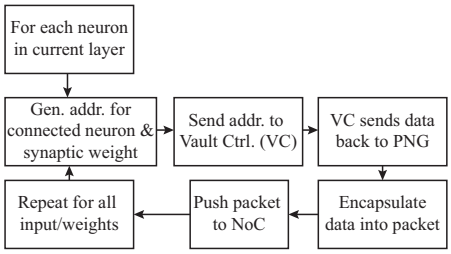

Figure 7. Operation of the Programmable Neurpsequence Generator (PNG) for computing one layer.

a MAC operation. Finally, if the size of synaptic weights matrix is small all weights are stored in PE weight memory.

C. 2D Network on Chip

The PEs are interconnected by a 2D mesh network as shown in Fig. 6 (a). Fig 6 (c) illustrates a block diagram of the router. Each PE is connected to a single router. Each router has 6 input channels and 6 output channels (4 for neighbhoring routers and 2 for PE and memory). The router is wormhole switched with credit-based flow control, a 16-depth packet buffer for each input and output channel, and table-based routing. Routing is deterministic X-Y routing. Input buffers use a rotating daisy chain priority scheme for arbitrating between inputs requesting the same outputs. Priorities are updated every clock cycle. The impact of the NoC architecture on the system performance in terms of throughput will be analyzed in Section VI.

IV. Memory Centric Neural Computing

In this section, we describe the design/programming of the Programmable Neuroseuquence Generator (PNG). The host programs the execution of one layer at a time. For example, Neurocube to operate ConvNN for scene labeling [9] with six layers (2Dconv, pooling, 2Dconv, pooling, 2Dconv, and fully connected) where different layers demonstrate different types of connectivity (local connection in 2Dconv, and all to all connectivity, similar to MLP in fully connected layer) should be programmed six times.

The execution of each layer is described in this section as i) the packetization and flow of data between memory and the PEs, ii) the addressing of memory and programming of the memory-based state machines, and iii) the programming interface to the host.

A. Orchestration of the Data Flows

Each vault controller in the HMC has an associated programmable neurosequence generator (PNG) that controls the data movements required for neural computation. Fig. 7 shows the operation of a PNG for each layer in the NN. For each neuron in the layer, the PNG will generate the addresses of the connected neurons in previous layer and weights in the memory. Consider a neuron in one layer (y_i in Fig. 8 (b)). To compute the state of this neuron the PNG generates a sequence of addresses for the locations of (a) the state of all connected neuron (x_k) and (b) the corresponding synaptic weights (w_k) between them. The preceding is repeated for each neuron in the network. The PNG executes this operation and sends the address sequences to the vault controller (VC).

As the PNG receives the data stream from the VC, the data corresponding to each connected neuron is encapsulated into a packet. The PNG encodes a specific MAC-ID for each packet such that all packets corresponding to the neuron and its connected neuron have the same MAC-ID. The PNG also pushes states (y_i in Eq. 1) through the non-linear activate function (implemented as the Look Up Table (LUT)) and the output of neuron (x_i in Eq. 2) is embedded in the packet. Finally, the packet includes source ID (memory vault ID) and destination ID (PE ID) and is injected into the router in the NoC to deliver to the PEs. (Fig. 8 (a)).

After the MACs finish the computation, the state of output neuron is encapsulated into packets for each MAC (all MACs finish the operation at the same time) with MAC$-$ID and injected into the network. When the PNG receives the packet (write back), the PE index (SRC in the packet and MAC-ID are sufficient to determine the neuron to be updated and its address. This information is pushed back to the VC.

B. Design and Operation of the PNG

Fig. 8 (a) shows the block diagram of the PNG, composed of i) the address generator, ii) configuration registers, iii) a Look-Up-Table (LUT) for the non-linear activate function, and iv) packet encapsulation/de-encapsulation logic. Each PNG is programmed by a global controller (μ ctrl.) interacting with the host. The host loads configuration registers (see below) to initiate the computation of a single layer. After all 16 PNGs are configured, computation begins.

The *address generator* in the PNG is designed as a programmable finite state machine (FSM) that can be used to sequence through addresses for single layer of neurons. Operationally, the computation over a single layer of neurons is composed of three nested loops: a loop across all neurons in the layer, a loop across all connections for single neuron in the layer, and a loop across all MACs. A single MAC computes the state of one neuron at a time; therefore n_{MAC} MACs compute n_{MAC} neurons after iterating over $n_{Connections}$ computations (multiplication and additions). This process is repeated until the state of all neurons in this layer have been computed. The FSM structure using three counters (one for each loop) is shown in Fig. 8 (b). These counters are programmed by the host to initiate the computation of each layer.

The *combinational logic* computes the memory address of each required connected neuron and synaptic weights for current neuron in this layer. This logic receives the current states of the *neuron counter* (cur_x, cur_y) and *connectivity*

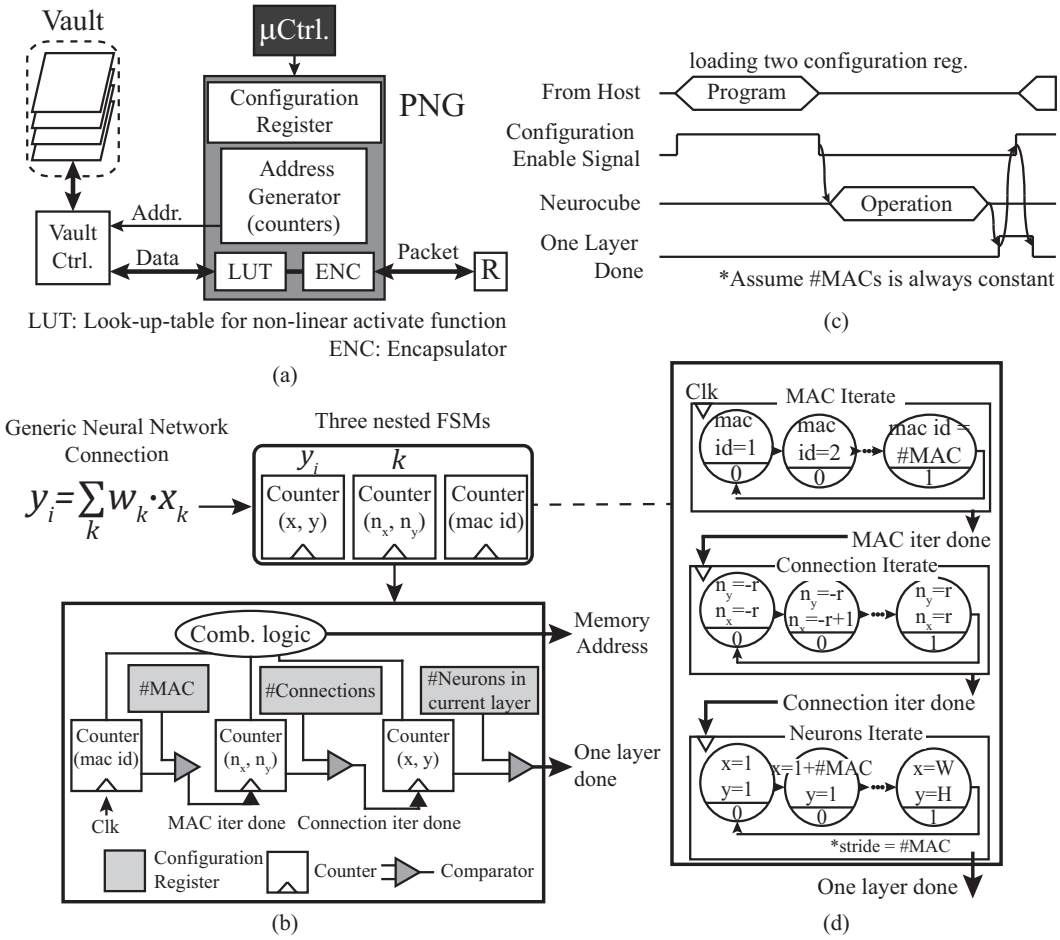

Figure 8. (a) Interaction between PNG, VC, and host. (b) General neural network can be translated as three nested loops, which can be implemented using three counters. (c) Timing diagram of programming PNG and Neurocube operation. (d) Three nested loops can be mapped to finite state machines.

(n_x, n_y), and computes the target address $(targ_x, targ_y)$ as follows:

$$targ_x = cur_x + n_x, \quad targ_y = cur_y + n_y, \tag{4}$$

Note (cur_x, cur_y) is the coordinate of the current neuron. Based on the the number of connected neurons and their weights, the memory range allocated for this layer (state of connected neurons and synaptic weights) can be precomputed. Therefore, the actual memory address for the required neuron located at $(targ_x, targ_y)$ is computed as:

$$Addr = targ_y \times W + targ_x + Addr_{last}, \tag{5}$$

where W represent width of output image and $Addr_{last}$ represents the last memory address from the previous layer.

Fig. 8 (c) shows timing of host-PNG interactions. To program the PNG, the host asserts a configuration enable signal to write configuration registers (see next section). After the host writes all configuration registers, it deactivates the configuration enable signal that initiates the FSM. Fig. 8 (d) shows that how the three counters inter-operate. When the state value of the *current neuron* counter equals the total number of neurons in this layer, it means that the PNG has generated all data address sequences required for this layer. After generating all required data addresses, the PNG waits to receive the newly computed state for the last output pixel. After it receives the last state, the PNG raises the layer done signal. Then host now starts programming the next layer.

C. Programming of the PNG

Programming the PNG initially map all data structures of NN (e.g., input image and weights) into the physical address space of the cube (i.e., across vaults, dies, banks etc.) followed by periodic configuration to process each layer of the neural net. The implementation of this global controller can be via software executing on a simple micro controller on the logic die or, directly by the host via the HMC links. In the latter case the configuration registers must be accessible to the host. In this paper we assume the latter, i.e., direct host programming.

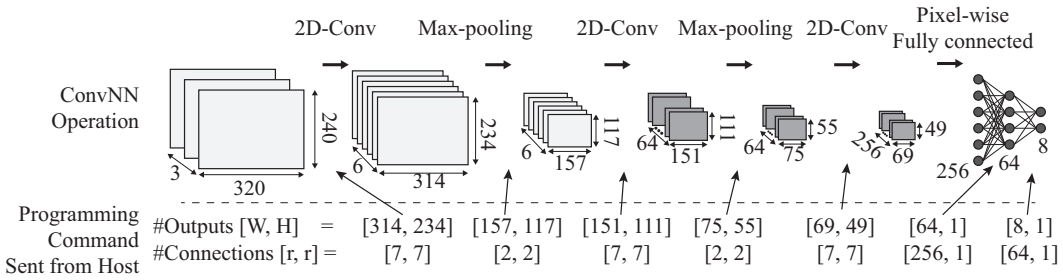

Figure 9. Convolutional neural network for scene labeling [9] and programming parameters for each layer

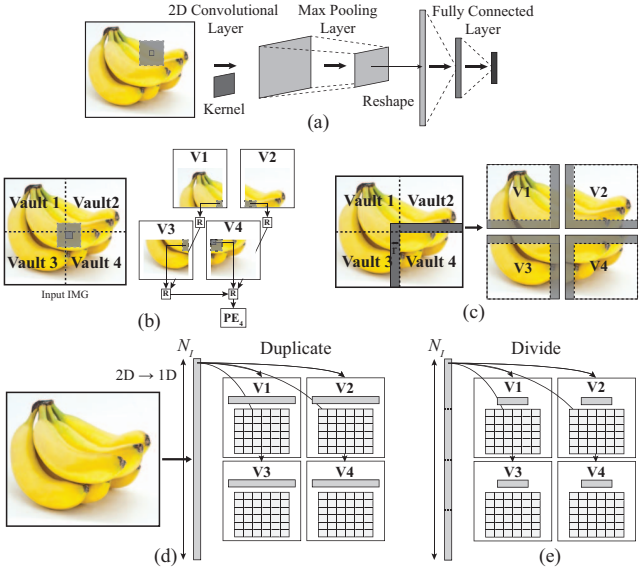

Figure 10. Data movement in Neurocube (assume 4 vaults and 4 PEs). (a) ConvNN structure. (b) For 2D convolutional layer, input image is divided into 4 non-overlap segments. (c) To reduce NoC traffic, input is divided with overlapped area. (d) For fully connected layer, input image is transformed to vector and this vector is duplicated to all HMC vaults. (e) To reduce duplicated memory overhead, input vector is divided into all vaults.

As an example, consider how the PNG can be configured for a convolutional NN during inference (Fig. 9). The number of MACs is determined by design as 16. To iterate over all neurons in the 1^{st} convolutional layer, the configuration register for the number of neurons should be set to 73,476 (314 $\times$ 234). The value of the counter for iterating over current neurons is incremented by 16 at each step since the states of 16 neurons in this layer are computed simultaneously. For one neuron, the number of connection is 49 (7 $\times$ 7), and is also programmed into the PNG.

V. Compute Operation in Neurocube

A. Management of Data Movement

Application: ConvNN for Scene Labeling

In this section, we discuss how to reduce data movement, specially over the 2D NoC, for locally connected and fully connected layer. Fig. 10 shows that how input image and weight matrix are divided into n_{Ch} partitions and stored in DRAM vaults to manage data movement.

1. Locally Connected Computation: For small connections, the weights are duplicated in the weight memory of all PEs. Only input needs to be partitioned and stored in the vaults as illustrated in Fig. 10 (b). The 2D NoC traffic can be reduced by dividing into multiple overlapped image segments (Fig. 10 (c)). The overlapping can improve throughput with small memory overhead, specially for small kernels; the memory overhead increases with kernel size.

2. Fully Connected Layer: We can divide the weight matrix into DRAM banks (Fig. 10 (d)) and duplicate the input vector in each vault so that a PE can compute neurons using data from a single vault. However, when input image is too large to duplicate, both input and weight matrix need to be divided resulting in higher NoC traffic (Fig. 10 (e)). Note a fully-connected model can be used to represent *irregular* connections between neurons by storing a synapse weight of '0' for missing connections.

B. Operation of the PEs

Fig. 11 illustrates PE operations after programming by global controller. Fig. 11 (a) shows data and packet transfer among vault, PNG, and router. From a single DRAM vault in HMC-Int., the PNG receives 32bit data and encapsulates that into two packets. Source (SRC) indicates the DRAM vault (4bit for 16 vaults in HMC-Int.), and destination (DST) indicates PE (4bit for 16 PEs). As all MACs operate in parallel, n_{MAC} weights and n_{MAC} states are delivered to PE. Each packet has a MAC-ID (4bit for n_{MAC}=16) to represent the target MAC. Each packet has OP-ID to represents the sequence of operations to compute one single output neuron. We assign 8bit for OP-ID. If maximum iteration for one pixel is more than 256, OP-ID represents the remainder of OP-ID divided by 256.

Fig. 11 (a)-(d) illustrates example when PE needs to compute 3^{rd} operation (OP-counter is 3). Packet for MAC_{15} to operate 3^{rd} operation is moved to temporal buffer[15] directly (a). If packet's OP-ID is higher than current OP-counter, it moves to cache memory (b). Cache memory is divided into multiple sub-banks (e.g. 16 sub-banks as in

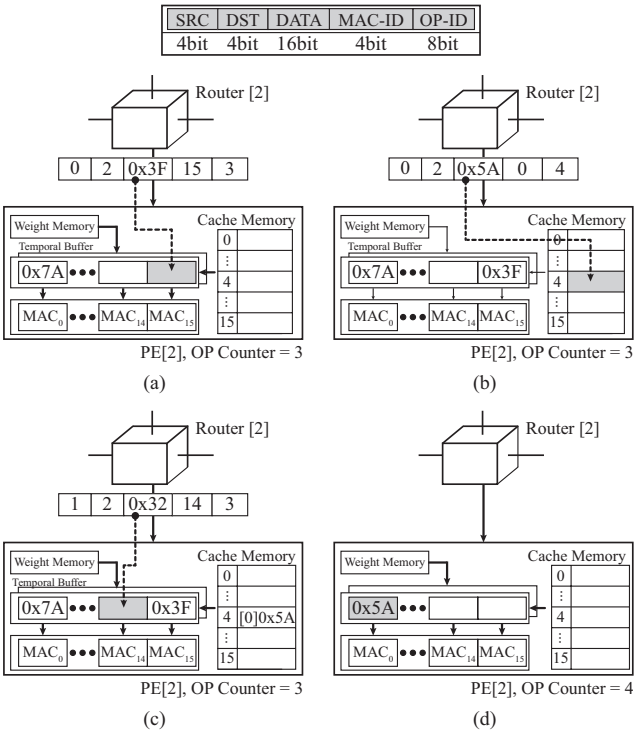

Figure 11. Operation of PE (OP counter = 3). (a) Packet with OP-ID as 3 arrives PE and moves to temporal buffer. (b) Packet with OP-ID as 4 arrives PE and moves to cache memory. (c) Packet with OP-ID as 3 arrives PE and moves to temporal buffer. Temporal buffer receives 16 weights and input (d) Buffer is flushed and MACs start computation. Operation counter increases. Before start 4^{th} operation, bring pre-stored data from cache memory.

Fig. 11). When a new packet arrives which is not for the current operation (OP-ID != OP-counter), it is stored in sub-bank mod(OP-ID, 16). When temporal buffer receives all 16 input pixels and 16 synaptic weights, temporal buffer is flushed, MACs start computation, and PE increases OP-counter (c). For next operation, pre-stored data in cache memory is moved to temporal buffer (d). When checking for necessary packets for the next operation, the PE performs a full search of the sub-bank that may take anywhere from 16 clock cycles (16 MACs) to 64 clock cycles (max 64 entries).

VI. System Throughput Simulation

We developed a cycle-level Neurocube simulator for performance evaluation. In simulator, main memory specification (bandwidth, number of channels, bus-width), PE (cache size, number of MACs), and router (buffer size, latency) are parametrized. For all 16 vaults in the HMC, 32bit word (2 data items) is pushed at 5GHz (HMC specification) in burst mode and burst length is assumed as 8. Therefore, after pushing 8 words, the HMC needs to wait t_{CCD} before sending the next 8 words. Reference clock in the simulator is the main memory clock frequency (2.5GHz $\times$ 2 = 5GHz).

The system simulation is performed considering a multi-layer ConvNN structure for scene labeling [9]. We choose ConvNN as the example, because it helps demonstrate the programmability of the Neurocube for locally connected (2D convolutional) as well as fully connected networks. This ConvNN is constructed with 7 layers and input image is RGB 320 $\times$ 240. The number of neurons for each layer is illustrated in Fig. 9. We analyze the system performance during both inference and training, while most of the prior hardware works only considered inference ([2] [4] [5] [8] [6]) or training for simple application like MNIST (2 layers, 28$\times$28 input, [7]).

Fig. 12 and Fig. 13 show the throughput and memory requirements for inference and training operations in scene labeling [9]. Unlike most of the related works that concentrates on only one layer and/or only inference operations, we simulate the system for *both inference and training* with and without data duplication.

1) Inference: The three convolutional layers and the first fully connected layer dominates the number of operations. In particular, the former dominates execution time in training. With data duplication (black bar), Neurocube shows almost constant throughput since there is no lateral traffic for both types of layers (132.4GOPs/s). Without data duplication (gray bar), throughput for fully connected layers degrades; therefore total throughput is slightly lower than that of data duplication (111.4GOPs/s).

2) Training: For training, operation with data duplication is applied (126.8GOPs/s). For memory requirements, duplication shows 48% additional memory overhead to maintain high throughput.

3) Image Processing Throughput: We estimate the image processing throughput for the scene labeling application based on the RTL level design of the Neurocube hardware in 28nm and 15nm nodes (see Section VII). The estimated throughput for inference are 17.52 frames/second in 28nm and 292.14 frames/second in 15nm. Likewise for training, 272.52 frames/second in 28nm and 4542.14 frames/second in 15nm (one epoch in training with image size: 64 $\times$ 64).

Extending Neurocube for Other Neural Networks: The example of ConvNN shows how Neurocube can be used to program other NN. Programming an MLP [13] or RNN [23] is similar to programming the fully-connected layer in ConvNN. Although RNN is not simulated in this paper, RNN is equivalent to a deep MLP after unfolding in time, while LSTM [28], a variant of RNN with multiple hidden layers each with a different activation function, can be realized by updating the LUT for each layer during programming. On the other hand, programming a locally connected layer like Cellular Neural Network [29] is similar to programming the 2D convolutional layer. Therefore, different types of network can be programmed in Neurocube without architectural changes.

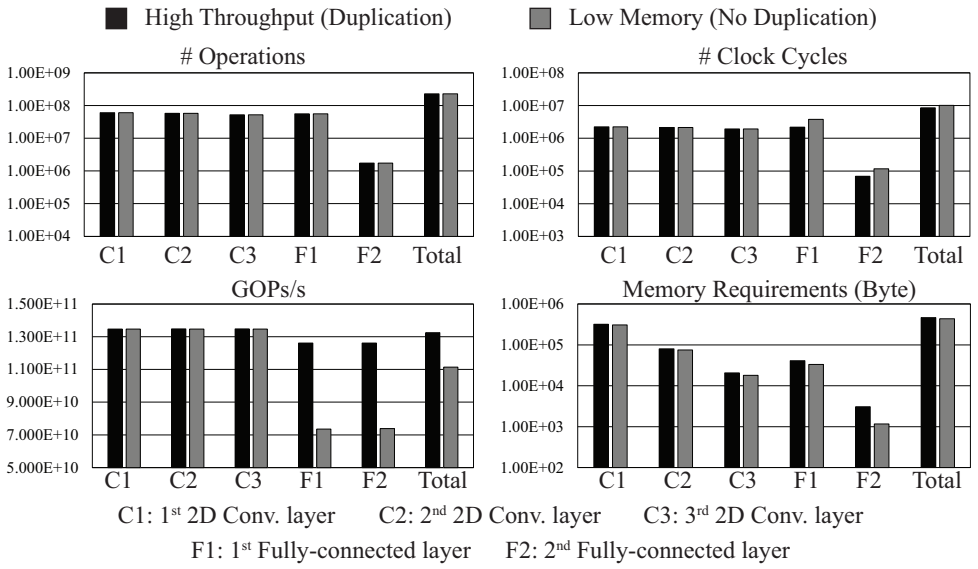

Figure 12. Neurocube performance for scene labeling [9]. Neurocube can operate high throughput with data duplication (black) or slightly lower throughput to save memory requirement (gray). (a) Number of operations, (b) Number of clock cycles, (c) Throughput (GOPs/s), and (d) memory requirements and overhead for inference with data duplication

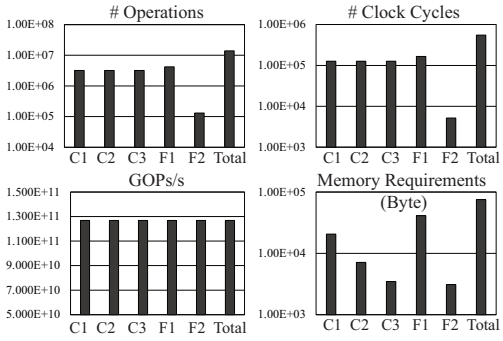

Figure 13. Neurocube performance for training scene labeling [9] with small image size (64 × 64) and data duplication. (a) Number of operations, (b) number of clock cycles, (c) throughput (GOPs/s), and (d) memory requirements and overhead for inference with data duplication

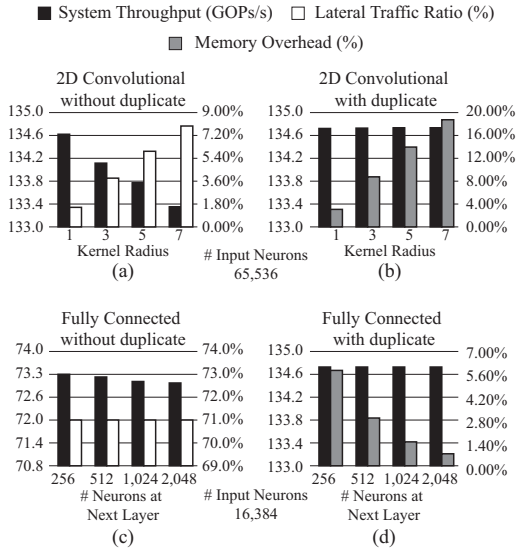

Figure 14. Effect of NN parameters on throughput and memory: effect of kernel size in 2D convolutional layer (a) without duplicate and (b) with duplicate; and effect of number of neurons at in the hidden layer for fully connected layer (c) without duplicate and (d) with duplicate.

A. *Effect of NN Parameters*

In this sub-section we study the effect of NN parameters on the system performance. Fig. 14 (a) and (b) show the simulation results of the locally connected layer (2D convolutional layer) with different kernel sizes. When each vault does not duplicate the neighborhood input pixels, a higher kernel size increases the lateral traffic on the NoC thereby decreasing system throughput. When the neighborhood is duplicated, there is no degradation in system performance with large kernel size, however, the memory overhead due to duplication increases.

Fully connected layers are placed as last layers of the ConvNN as classifiers. Consider a 3-layer fully-connected network with one hidden layer (between input and output). More neurons in the hidden layer allow more complex non-linear classification, but also require more computation and memory. Without duplication, one PE should access all vaults to compute one pixel at the current layer; therefore lateral traffic on the NoC is high (71%) (Fig. 14 (c)). However, the number of connected neurons in previous layer for one neuron is constant. In other words, the amount of lateral traffic is also constant. Therefore, system performance

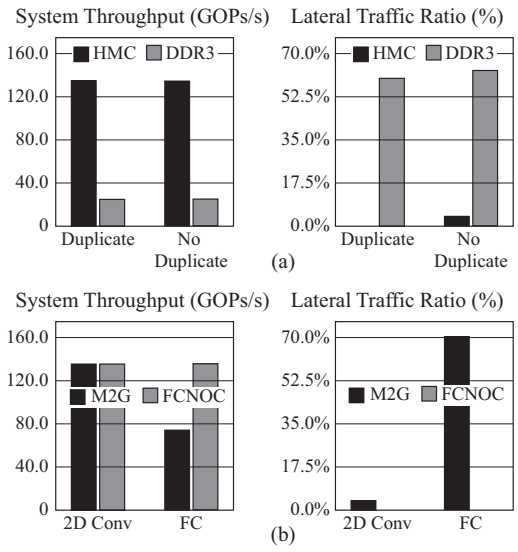

Figure 15. Performance comparison: (a) HMC and DDR3, and (b) mesh grid and fully connected NoC.

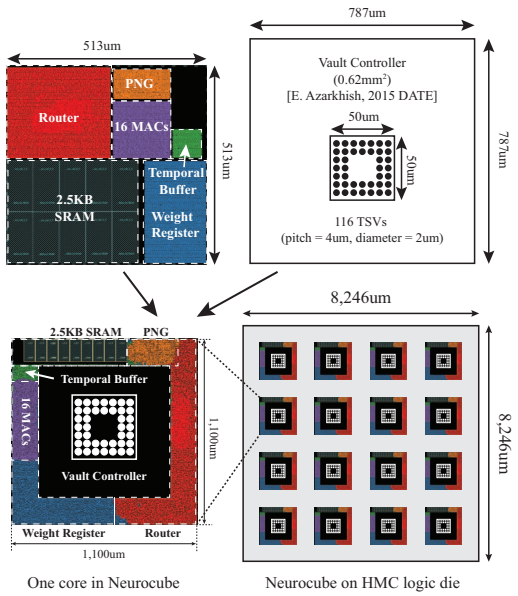

Figure 16. Layout of one partition of HMC logic die including a single PE (16 MACs, 2.5KB SRAM, weight registers, temporal buffer, and PNG), vault controller [24], and a single router.

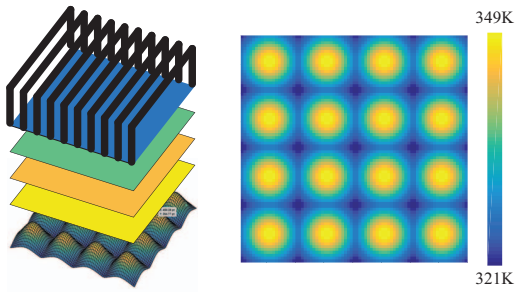

Figure 17. 3D-Thermal simulation for Neurocube in 15nm and 4 DRAM dies. Maximum temperature of logic die is 349K and of DRAM die is 344K.

is almost constant (Fig. 14 (d)). When number of neurons at the current layer increases, synaptic weights ($N_I \times N_O$) occupy the most of the memory in terms of space since it is fully connected (it has a huge weight matrix). Therefore, the portion of duplicated input neurons in total required memory decreases (memory overhead decreases).

B. HMC-Internal vs. DDR3

According to Table I, peak bandwidth of DDR3 (12.8GBps) is higher than that of HMC-Int (10GBps); therefore, a comparative analysis of the two will help understand the role of concurrency in the Neurocube. For the 2D Conv. layer, simulation is performed to analyze the impact of the number of channels. Fig. 15 (a) shows that DDR3 shows much lower system performance, although DDR3 has higher peak bandwidth. Since DDR3 has only two channels, data traffic on the 2D NoC is a major bottleneck. Due to large lateral NoC traffic (about 60%), there is no benefit from duplication in 2D Conv. in terms of system performance. Under the same bandwidth, more slower channels can leverage NoC traffic; therefore it improves the system performance.

C. Mesh Grid NoC vs. Fully Connected NoC

In addition to memory with many channels, a fully connected NoC (all routers are connected each other, Fig. 6 (b)) also can reduce the NoC traffic. The impact of NoC on the performance is emphasized especially for the layer with dense connections. Fig. 15 (b) shows that there is no throughput degradation from the locally connected layer to the fully connected layer since there is no lateral traffic on the NoC. However, one single router needs 17 input/output channels. High radix NoCs may be an option here.

VII. Hardware Simulation

Design of PE: One PE was designed in Verilog and synthesized using 28nm CMOS and 15nm FinFet [31], [32]. The PE includes 16 MAC units, PNG, temporal buffer, and weight memory. As already explained in Section III-B, 2.5KB local memory is composed of 16 banks and each bank is designed to store 1,280bits (80bit $\times$ 16 lines) : 20bit word (16bit data + 4bit MAC-ID) $\times$ 16 MACs $\times$ 4 buffering depth. For each vault, we design one PE and a router for 2D mesh. The maximum clock frequency of SRAM from the 28nm library was 300MHz. Therefore, the PE and the router were synthesized to operate at 300 MHz and the MACs are synthesized to operate at 18.75MHz (Eq. 3). We have also re-designed the Neurocube with 15nm FinFet process [32] to achieve 5GHz operating frequency. Power and area of

Table II
HARDWARE SIMULATION OF SINGLE CORE IN NEUROCUBE

		Operating Freq (MHz)		Dynamic Power (W)		Area (mm^2)		Power Density (W/mm^2)	
	Size (bit)	28nm	15nm	28nm	15nm	28nm	15nm	28nm	15nm
MAC	16	18.75	320	3.02E-04	9.17E-03	0.0011	0.0002	2.75E-01	4.89E+01
SRAM Cache (2.5KB)	20,480	300	5,120	2.93E-03	2.90E-02	0.0873	0.0448	3.36E-02	7.10E-02
Temporal Buffer	512	300	5,120	2.70E-05	2.05E-05	0.0025	0.0003	1.09E-02	7.10E-02
PMC	N/A	300	5,120	4.17E-04	1.39E-03	0.0081	0.0013	5.16E-02	1.09E+00
Weight Reg	3,600	300	5,120	1.84E-04	1.44E-04	0.0173	0.0020	1.07E-02	7.10E-02
Router	36	300	5,120	7.17E-03	3.59E-02	0.0609	0.0085	1.18E-01	4.24E+00
PE Sum	-	300	5,120	1.56E-02	2.13E-01	0.1936	0.0600	8.04E-02	3.06E+00
Compute in Neurocube (16 PEs + 16 Routers)	-	300	5,120	2.49E-01	3.41E+00	3.0983	0.9601	8.04E-02	3.06E+00
HMC Logic Die Without Neurocube	-	300	5,120	1.04E+00	8.67E+00	-	-	-	-
All DRAM Dies	-	300	5,120	5.68E-01	9.47E+00	-	-	-	-

Table III
RECENT HARDWARE PLATFORMS FOR NEURO-INSPIRED ALGORITHM.

Papers		[2] '15		This work		[4] '11		[5] '14	[7] '14	[8] '15	[6] '15
Programmability		Yes	Yes	Yes	Yes	No	No	No	No	No	No
Hardware		Tegra K1	GTX 780†	28nm	15nm	Virtex 6	45nm	Xilinx ZC706	28nm	65nm	28nm
Bit Precision		N/A	N/A	16bit	16bit	16bit	16bit	16bit	16bit	12bit	16bit
Thrp. (GOPs/s)	With DRAM	76.0	1,781.0	8.0	132.4	-	-	227.0	-	-	-
	W/O. DRAM	-	-	-	-	147.0	1,164.0	-	5,580.0	203.0	2.78
Compute Power (W)		11.00	206.80	0.25 (1.86)	3.41 (21.50)	10.00	5.00	8.00	15.97	1.20	0.001
Efficiency (GOPS/s/W)		6.91	8.61	31.92	38.82	14.70	232.80	28.38	349.40	169.17	2,780.00
Application Inference/Training		Scene Labeling [9] Inference		[9] Both		N/A	N/A	N/A	N/A Both	[9] Inference	N/A
Number of Input Neurons		76,800		76,800		N/A	N/A	N/A	784	76,800	N/A

† GDDR5 (3GB, 6Gbps) power is estimated by [30]

SRAM in 15nm is estimated using [11] (power). Supply voltage ratio is used to estimate SRAM power in 15nm, which is conservative estimation.

Power estimation of HMC: The baseline HMC design reports 3.7pJ/bit for DRAM and 6.78pJ/bit for logic layer [20]. Based on these values, the power of the logic die without Neurocube (16 vault controllers, 4 links (SERDES), interface between all VCs and all links) and DRAM die is computed assuming clock frequency of the vault I/O clock (2.5GHz $\times$ 2 = 5GHz) [i.e. Logic power = 6.78pj/bit $\times$ 32 bit $\times$ 16 $\times$ 5GHz = 17.3 W]. However, the maximum clock frequency for the PE in the 28nm node is only 300MHz, leading to a reduced activity of 0.06 (=300MHz/5GHz) in the vault controllers and DRAM. Hence, logic die and DRAM powers in 28nm are scaled accordingly. As the PE in 15nm operates at 5GHz, no such activity scaling has been applied. However, the baseline power of the logic die have been scaled based on the energy scaling factors from [33].

System power and performance: Table II shows the dynamic power consumption and area for Neurocube in 28nm and 15nm nodes. Therefore, additional power overhead due to the Neurocube on the logic die is 249mW (16 $\times$ 15.6mW) in 28nm and 3.41W in 15nm. The image processing throughput for inference and training using Neurocube are mentioned in Section VI.

Area analysis: The area overhead in the logic die due to 16 PEs is 3.09mm^2 (16 $\times$ 0.1936mm^2) in 28nm and 0.98mm^2 in 15nm. To estimate total logic die area with Neurocube, we present one feasible layout in 28nm (Fig. 16). One PE and a router can be placed in 513μm^2 by 513μm^2 with 70% utilization ratio. We used area of the VC synthesized in 28nm from [24]. As there are 1,866 TSVs in one HMC [20], we assumed that 116 TSVs are placed in the middle of the VC and the area of the TSV array is estimated using a 4μm pitch and 2μm diameter [33]. One core of Neurocube (a PE, a router, and a VC) is designed by placing VC in the middle and placing other modules around the VC to reduce interconnect. We can see that Neurocube with 16 cores can be synthesized on the logic die (68mm^2 [20]) of HMC. Neurocube in 15nm also fits in the area of HMC (Table II).

Thermal analysis: For thermal analysis of Neurocube we use [34], [35] and simulate the floorplan shown in

Fig. 16 assuming passive heat sink. For the 28nm node, the thermal effect was negligible as Neurocube consumes relatively small power at 300MHz (1.3W). For the 15nm node (and associated power density), We observe that the maximum temperature for 16 PEs increases up to 349K and the maximum temperature for 4 DRAM dies increases to 344K (Fig. 17). According to the HMC 2.0 [36], the maximum operating temperature of logic die is 383K and that of DRAM die is 378K. Therefore, Neurocube operating at 5GHz at the 15nm node fits within thermal conditions.

We estimate logic die power based on [20], which includes the power of 16 vault controllers, 4 high speed links, ECC, and the interface between vaults and links. Note in Neurocube we will use links, ECC, and interface only during programming by host, not during computation; this may reduce the logic die power from Table II.

VIII. Related Work

The related work on the neuromorphic hardware platforms can be divided into two categories: software based on GPUs, and hardware based using ASIC/FPGA (Table III).

GPU [2] and mobile processor [2] can operate multiple different neural networks by programming software and interface with external high density memory. However, GPUs have lower compute efficiency than ASIC/FPGA.

ASIC/FPGA platforms can achieve higher compute efficiency. FPGA platforms for the 2D convolutional layers have been demonstrated [3]–[5]. The designs leveraged localized computations on ConvNN by reusing neuron's output to improve efficiency; however, it may not work on fully connected layers. Note even in FPGA, once a NN engine is synthesized, it cannot be programmed on-line.

ASIC can achieve the highest throughput and the highest efficiency, but its scalability is limited by on-chip memory (or interface with external memory) and lack of programmability [4], [6]–[8]. However, including the DRAM latency and power will degrade efficiency of these engines.

The Neurocube presented in this paper takes a significant shift from the prior work. First, Neurocube explores the processing-in-memory concepts to address the critical memory capacity/bandwidth challenge allowing a scalable NN hardware with high system performance. More importantly, Neurocube can be programmed on-line to map different neural networks (programmability). As expected, Neurocube shows higher compute efficiency than GPU (including DRAM latency). The ASICs show better performance, however, a part of the difference can be attributed to the fact prior work did not consider the DRAM performance.

IX. Conclusion

This paper presents the Neurocube - a programmable and scalable digital architecture for neuro-inspired algorithms. The Neurocube design incorporates a logic die with an array of clustered, data-driven, multiply-accumulate (MAC) units integrated with a 3D DRAM stack for high bandwidth and low latency memory access. Programmable address sequence generators integrated into the memory system generate the correct sequence of data accesses to push data from memory to the MAC units where the arrival of neuron states and connectivity weights triggers MAC operation. By reprogramming the sequence generators, multiple, different types of neural networks can be emulated. Next steps involve scaling this implementation across multiple cubes to support much larger networks than can be feasibly supported today.

Acknowledgment

This material is based on work supported in part by an ONR Young Investigator award, a National Science Foundation CAREER Award, and the National Science Foundation grant CCF 1337177. The authors also gratefully acknowledge the detailed and constructive comments of the reviewers.

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[2] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 108, ACM, 2015.

[3] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An fpga-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 32–37, IEEE, 2009.

[4] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pp. 109–116, IEEE, 2011.

[5] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pp. 696–701, IEEE, 2014.

[6] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 683–688, EDA Consortium, 2015.

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

[8] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 199–204, ACM, 2015.

[9] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1–8, IEEE, 2009.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] E. Karl, Z. Guo, J. W. Conary, J. L. Miller, Y.-G. Ng, S. Nalam, D. Kim, J. Keane, U. Bhattacharya, and K. Zhang, "0.6 v 1.5 ghz 84mb sram design in 14nm finfet cmos technology," in *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pp. 1–3, IEEE, 2015.

[12] F. Hamzaoglu, U. Arslan, N. Bisnik, S. Ghosh, M. B. Lal, N. Lindert, M. Meterelliyoz, R. B. Osborne, J. Park, S. Tomishima, *et al.*, "13.1 a 1gb 2ghz embedded dram in 22nm tri-gate cmos technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 230–231, IEEE, 2014.

[13] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," tech. rep., DTIC Document, 1961.

[14] "DDR3 SDRAM, JESD79-3F." http://www.jedec.org/standards-documents/docs/jesd-79-3d.

[15] "WIDE I/O 2, JESD229-2." http://www.jedec.org/standards-documents/docs/jesd229-2.

[16] "High Bandwith Memory, JESD235." http://www.jedec.org/standards-documents/results/jesd235.

[17] Hybrid Memory Cube Consortium, "Hybrid memory cube specification 1.0," 2013.

[18] P. Rosenfeld, *Performance exploration of the hybrid memory cube*. PhD thesis, University of Maryland, 2014.

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

[20] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *2012 Symposium on VLSI Technology (VLSIT)*, 2012.

[21] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," in *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 37–48, IEEE Computer Society, 2012.

[22] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*, vol. 3. Pearson Education Upper Saddle River, 2009.

[23] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. German National Research Center for Information Technology, 2002.

[24] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "High performance axi-4.0 based interconnect for extensible smart memory cubes," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1317–1322, EDA Consortium, 2015.

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

[26] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 336–348, ACM, 2015.

[27] J. Zhao, G. Sun, G. H. Loh, and Y. Xie, "Optimizing gpu energy efficiency with 3d die-stacking graphics memory and reconfigurable memory interface," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, p. 24, 2013.

[28] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[29] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *Circuits and Systems, IEEE Transactions on*, vol. 35, no. 10, pp. 1257–1272, 1988.

[30] "Samsung Green Memory Solutions." http://www.samsung.com/us/business/oem-solutions/pdfs/Green-GDDR5.pdf.

[31] "Synopsys 32/28nm Generic Library." https://www.synopsys.com/COMMUNITY/UNIVERSITYPROGRAM/Pages/32-28nm-generic-library.aspx.

[32] "Nangate FreePDK15 Open Cell Library." http://www.nangate.com/?page_id=2328.

[33] "International Technology Roadmap for Semiconductors, Interconnect, 2011." http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20Links/2011ITRS/2011Chapters/2011Interconnect.pdf.

[34] A. Sridhar, A. Vincenzi, D. Atienza, and T. Brunschwiler, "3d-ice: A compact thermal model for early-stage design of liquid-cooled ics," *Computers, IEEE Transactions on*, vol. 63, no. 10, pp. 2576–2589, 2014.

[35] W. J. Song, S. Mukhopadhyay, and S. Yalamanchili, "Energy introspector: A parallel, composable framework for integrated power-reliability-thermal modeling for multicore architectures," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pp. 143–144, IEEE, 2014.

[36] Hybrid Memory Cube Consortium, "Hybrid memory cube specification 2.0," 2014.