aCortex: An Energy-Efficient Multi-Purpose Mixed-Signal Inference Accelerator

Mohammad Bavandpour, Mohammad R. Mahmoodi, and Dmitri B. Strukov, *Senior Member, IEEE*

***Abstract*—We introduce "aCortex", an extremely energy efficient, fast, compact, and versatile neuromorphic processor architecture suitable for acceleration of a wide range of neural network inference models. The most important feature of our processor is a configurable mixed-signal computing array of vector-by-matrix multiplier (VMM) blocks utilizing embedded nonvolatile memory arrays for storing weight matrices. Analog peripheral circuitry for data conversion and high-voltage programming are shared among a large array of VMM blocks to facilitate compact and energy-efficient analog-domain VMM operation of different types of neural network layers. Other unique features of aCortex include configurable chain of buffers and data buses, a simple and efficient instruction set architecture and its corresponding multi-agent controller, programmable quantization range, and a customized refresh-free embedded dynamic random-access memory. The energy-optimal aCortex with 4-bit analog computing precision was designed in 55 nm process with embedded NOR flash memory. Its physical performance was evaluated using experimental data from testing individual circuit elements and physical layout of key components for several common benchmarks, namely Inception-v1 and ResNet-152, two state-of-the-art deep feedforward networks for image classification, and GNTM, a Google's deep recurrent network for language translation. The system level simulation results for these benchmarks show energy efficiency of 97, 106, and 336 TOp/J, respectively, combined with up to 15 TOp/s computing throughput and 0.27 MB/mm^2 storage efficiency. Such estimated performance results compare favorably with those of previously reported mixed-signal accelerators based on much less mature aggressively scaled resistive switching memories.**

Index Terms — Artificial Neural Networks, Neuromorphic Inference Accelerator, Mixed-Signal Circuits, Nonvolatile Memory, Floating-Gate Memory, Machine Learning

I. Introduction

The rapidly growing range of applications of machine learning for image classification, speech recognition, and natural language processing along with maturing of the neural network algorithms, especially for deep learning, have led to an urgent need in a specialized neuromorphic hardware [1-3]. At least for the next several years, the demand for fast, low-precision inference accelerators will remain higher than for higher-precision systems for network training, as projected by NVidia Corp., a leading company in the machine learning hardware [4].

The vast majority of the proposed neuromorphic accelerators from industry and academia are digital [5-8] – see also extensive review of various proposals in [2]. The most natural approaches, however, are based on analog and mixed-signal circuits [9-30]. Though the core principles of analog computing had been developed almost four decades ago [9, 10], its efficient implementations were enabled only recently [14-30] due to the emergence of novel continuous-state, nonvolatile, memory devices [31, 32]. Such memories enable very dense implementation of weights and of in-memory computing for vector-by-matrix multiplication, the most common operation in machine learning. Among different candidates, the resistive switching memories, including phase change and conductive bridge memories, metal-oxide memristors (also known as ReRAM or RRAM [31]) are perhaps the most promising due to their excellent scaling prospects. Their technology, however, is still in need of improvement, which is less of a problem for another excellent candidate, floating gate (FG) memories, e.g. those based on redesigned commercial-grade embedded NOR flash [22, 32, 33]. Though planar FG cells are less dense than passively integrated memristors, their main advantage is FG cell amplification, which simplifies and reduces overhead of peripheral circuitry. It is worth noting that the limited endurance of memristors and FG memories is less of an issue for inference applications, since the weights are typically reprogrammed infrequently.

In this paper, we present a multi-purpose inference accelerator, dubbed "aCortex", that is designed to capitalize on in-memory mixed-signal computing with nonvolatile memories. Though the idea of employing mixed-signal VMM based on nonvolatile memories for multi-purpose inference accelerators is not new [23-27], our work is novel in several aspects. Its key advantage is more extensive use of analog computing, not only for VMM computation but also for data transfer. Such approach minimizes the area/energy/delay overhead of the sensing and data conversion peripheries which are key factors limiting the efficiency of the mixed-signal neural accelerators [23-30]. A more compact design, in turn, reduces communication overhead due to shorter distances for data transfer. Moreover, data transfer overhead is further reduced by using a configurable chain of buffers exploiting the data reuse for convolution operation, programmable data buses that can be efficiently tailored on the fly to a particular utilization of mixed-signal array, and a custom-designed refresh-free embedded dynamic random-access memory (eDRAM) tailored to meet the retention time requirement. We also propose a simple and efficient instruction set architecture (ISA) along with a multi-agent controller, which takes advantage of the eligible time-overlap between consecutive micro-operations while minimizing the instruction memory (IM) requirement. Finally, we developed a system-level estimator which imports the target network's computational graph along with experimental and circuit-level simulation results for different architecture components, including digital-

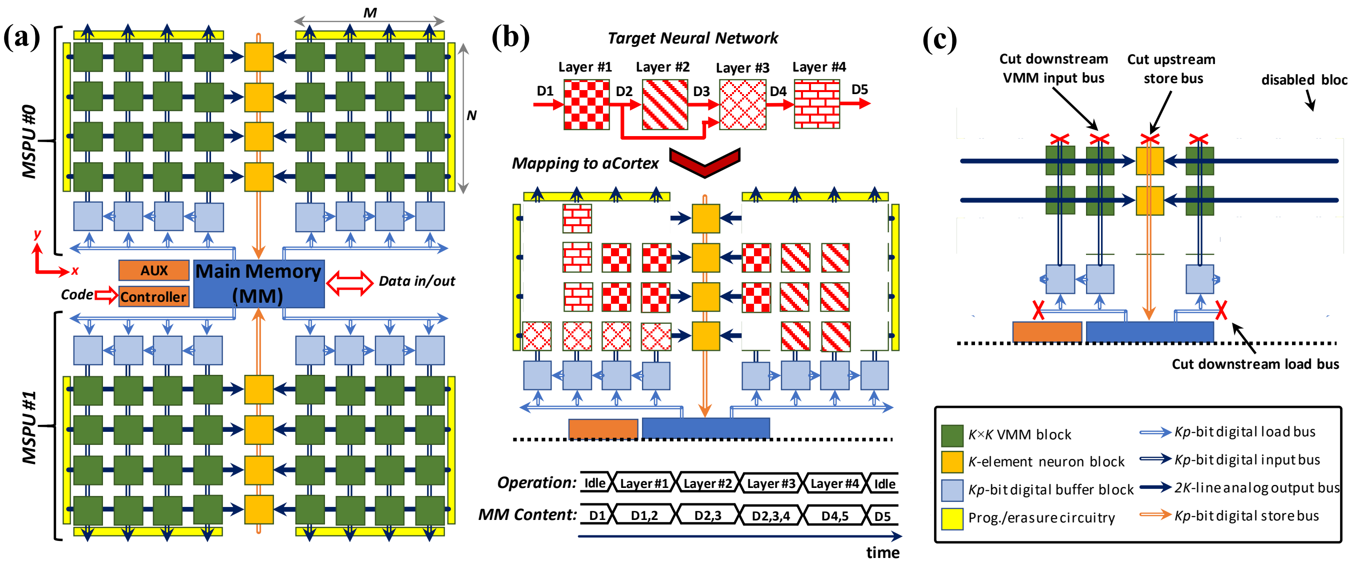


Fig. 1. (a) aCortex top-level architecture. The location of the components crudely corresponds to the actual layout and is chosen to reduce data transfer overhead. For clarity, the architecture is shown for $N = M = 4$ and most of the control lines and the circuitry for testing / weight tuning are omitted. (b) Example of a weight kernel mapping on aCortex VMM blocks, layer-by-layer operation scheduling, and corresponding content of main memory over time. (c) aCortex active blocks/bus portions during the execution of the layer #1 of the neural network shown in (b).

to-analog converters (DACs), analog-to-digital converters (ADCs), sense amplifiers, memory cells, digital blocks, and buses, maps the weight kernels onto the two-dimensional (2D) array of nonvolatile memory (NVM) blocks, and finally produces a comprehensive performance report considering various non-idealities such as leakages and line parasitics. Using such simulator, we perform a detailed performance analysis based on the actual layout in 55-nm process with embedded NOR flash memory. Note that unlike many proposals based on emerging memory technologies, core components have been previously taped out using commercial processes and successfully tested and we used such experimental data in our analysis.

In Supplementary Information (SI) Section S.I we introduce today's major neural layer types and present their hardware-friendly re-arrangement targeting a weight-stationary implementation. The overall aCortex architecture and operation scheme, as well as the internal structure of its main components are presented in Section II. This section also introduces the proposed instruction set architecture (ISA) along with the controller architecture. More details on the ISA are provided in Section S.II. The general framework for mapping applications into aCortex and our case study for three representative neural network inference tasks are provided in Section III. In Section IV, we provide the circuit diagram and experimental/simulation results for FG-based implementation of aCortex's core computing units in 55-nm technology node. We then perform a design space exploration for architectural parameters and provide a detailed system-level report for a semi-optimal design point. Related prior works are discussed and compared to aCortex in Section S.III. The paper is concluded in Section V, where important future works are outlined.

II. aCortex Architecture

A. Top-Level Architecture

As shown in Fig. 1a, the major processor's components are auxiliary unit (AUX), microcontroller, main memory, and two mixed-signal processing units (MSPU). Each MSPU includes a configurable chain of input digital buffer blocks, a flexible 2D array of VMM blocks, and an array of output neuron blocks. The architecture can be loosely characterized as Harvard weight-major type [2]. The instructions are stored in microcontroller's dedicated SRAM-based instruction memory. All frequently changing data, i.e. input, output, and temporary data, are kept in eDRAM-based main memory, while fixed weights, which would be typically precomputed at ex-situ training, are stored in NVM arrays of MSPU's VMM blocks.

The inference task is specified by a program code based on custom instructions and corresponding set of neural layer weight matrices. Assuming the code is loaded, and all weights are set up accordingly, the inference is computed by loading input data to the main memory, executing a code to perform the inference task, and storing the computed results back in the main memory. In particular, the stationary weight matrices corresponding to various network layers are packed in the 2D array of VMM blocks (Fig. 1b), and the inference is performed in a layer-by-layer manner by sequentially reading the layer input from the main memory into the digital buffer blocks, activating the appropriate VMM and neuron blocks to perform the target neural layer, and then temporarily storing the intermediate results in the main memory for computing the next layer. Note that some of the neural layers, such as CNV and LSTM require multiple VMM operations with various input data on the same weight matrix to complete. In this case, the corresponding VMM/neuron blocks are activated multiple times during the execution of each neural network layer - more details on that are provided in Sections III and S.I below.

Flexible activation of VMM/neuron blocks enables compact implementation of a set of neural network layers with various VMM sizes while maximizing the energy efficiency by cutting off the active power consumption of unutilized VMM/neuron blocks. Moreover, aCortex minimizes the energy overhead of data transfer by cutting off the unutilized portion of data buses and effectively reducing their length via disabling further data propagation. For example, Fig. 1c shows the active blocks and

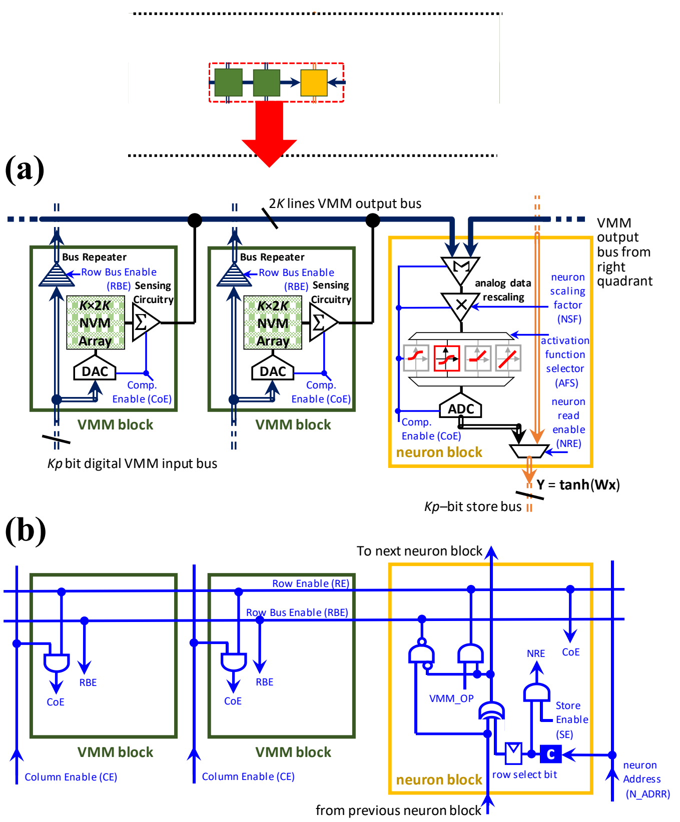


Fig. 2. (a) Schematic diagram of mixed-signal VMM and neuron blocks including their connectivity and required control signals (blue). Neuron block includes four stages, namely global sensing, re-scaling unit, activation function, and ADC. (b) The control circuitry, which facilitates flexible activation of the target VMM/neuron block, and cuts unutilized portions of the VMM input and store buses. Square labeled with 'c' denotes comparator for selecting a specific neuron. For clarity, some of the details, e.g. circuitry for setting up enable bits and some neuron control circuitry, are not shown.

buses when the processor is computing the first layer of the neural network shown in Fig. 1b.

All inference data, i.e. inputs and outputs of the MSPU's VMM blocks, as well as effective weight precision, are p bit. A set of p bits is defined as one data *word*. We specifically consider $p = 4$, which is typically sufficient for running state-of-the-art image classification inference without loss of functional performance [34, 35]. Blocks and buses are sized according to a global architecture parameter K, defined as the granularity of neural computation and data transfer on aCortex.

B. Main memory

Main memory is implemented with eDRAM technology, whose retention time is tailored for a refresh-free operation. In particular, it is organized as an array of eDRAM blocks, each with Kp-bit I/O data port. The multi-banked structure allows to read and write K data words simultaneously, i.e. to supply data to one digital buffer block and receive data from one neuron block. The required memory capacity and its retention time are calculated by monitoring the memory content and the longest lifetime of intermediate data for the inference of the target neural networks (Fig. 1b).

C. Mixed-Signal Processing Unit: Array of VMM Blocks

Each MSPU is comprised of two N-by-M arrays (quadrants) of VMM circuit blocks located on each side of a column with N neuron blocks. Each VMM block features K-by-$2K$ array of NVM cells, which is suitable for implementing analog-mode differential K-by-K VMM operation; K p-bit front-end DACs; and $2K$ back-end local sensing circuitry (Fig. 2a). The data to a single column of VMM block array is fed via Kp-bit wide digital programmable "VMM input" bus from the corresponding distributed memory buffer block. The VMM block outputs are connected via *analog* "VMM output" buses, which are $2K$ lines wide, to the corresponding neuron blocks. More circuit details on the data conversion and sensing/summation for the considered VMM design based on 2D-NOR flash memory technology is provided in Section IV-A.

Column/row enable lines (denotes as CE/RE) span the MSPU quadrants in vertical/ horizontal directions and are used to activate the desired VMM blocks at each processing step. Specifically, a given VMM block is activated only if its compute enable signal ($\mathrm{CoE} = \mathrm{CE} \wedge \mathrm{RE}$) is equal to "1" (Fig. 2b). These control lines allow to flexibly implement a wide range of VMM sizes (from $K{\times}K$ to $2MK{\times}NK$) while cutting off the active power consumption of unutilized VMM/neuron blocks. Moreover, a Kp-bit wide VMM input bus repeater with horizontally shared enable control line (RBE) is integrated in each VMM block (Fig. 2a) to speed up data propagation as well as to minimize its energy overhead by cutting the unutilized portion of the VMM input bus. The logic circuitry and single-bit registers producing CE and RE/RBE control lines are integrated in digital buffer and neuron blocks, respectively.

The programming/erasure circuitry consists of decoders and level-shifters, which are shared by NVM arrays of VMM blocks and are placed at the outer margins of the VMM block arrays (Fig. 1a), and row/column access switches, placed at the periphery of each VMM block, controlling the applied signals.

D. Mixed-Signal Processing Unit: Neuron Block

A neuron block includes K identical neuron units. All units perform in parallel summation/integration of the analog data supplied from the corresponding VMM output lines, re-scale the integrated data (if needed), apply the selected activation function, and finally convert the results to digital domain using ADCs (Fig. 2a). The rescaling unit enables a wide range of quantization ranges, e.g. needed to operate with different VMM input sizes, via adjusting neuron's analog input amplitude to match the fixed operating input range of the activation functions and ADC units. The activation function can be selected from linear, rectified linear, sigmoid, or hyperbolic tangent types, to support a wide range of neural layers (Section S.I). Neuron block outputs are digitized with p-bit ADCs and temporarily latched in their embedded digital memory. The digital results are then passed via Kp–bit wide digital "store" bus to the main memory. Such data transfer, i.e. the "store" operation, is performed in one step per neuron block, so that, e.g., a total of N steps is required to transfer data from all neurons in one MSPU. The specific number of processor cycles required for each step varies based on the location of the neuron block, i.e. its distance from the main memory. In particular, the store bus data are passed via a Kp-bit 2-to-1 multiplexer in each neuron block. These multiplexers act as a bus repeater for the utilized portion of the store bus, and is also used to decouple and deactivate its unutilized portion.

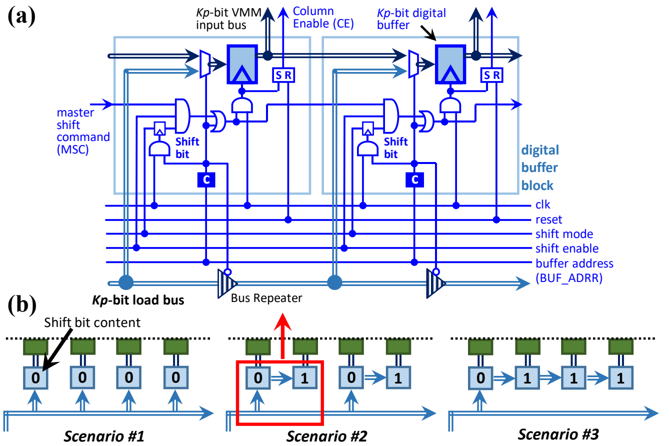


Fig. 3. (a) A detailed circuit implementation of the digital buffer block (DBB). Control circuitry is shown with blue color. Square labeled with ‘c’ denotes comparator for selecting a specific DBB. For clarity, some of the details are not shown. (b) Examples of three DBB chain configurations scenarios. Specifically, scenario #1 shows loading (sequentially) data exclusively from the load bus. Scenarios #2, 3 show loading data from load bus into DBBs and simultaneously shifting data between DBBs connected in a two-block shift register (#2) or a four-block shift register configuration (#3).

The RE and RBE signals (which control the target neuron block and the corresponding VMM blocks it serves, and cut the VMM input bus, respectively) are configured with “row-select” single-bit flip-flops of the neuron blocks (Fig. 2b). Specifically, the row-select flip-flops of neuron blocks are connected via XOR gates, with output of XOR gate directly controlling RE line. To preselect RE signals for the contiguous set of rows (e.g. rows 10 through 15), row-select bits of neuron blocks in the first (10), closest to the main memory, and in the last+1 (16) row of the selected set of rows are set to “1”, while others are set to “0”. This implementation results in activating all selected rows (i.e. setting RE = 1 for rows 10 through 15). On the other hand, using a simple NAND gate which detects a transition from selected to unselected row, RBE is de-asserted only for the last+1 row of the selected group of rows (Fig. 2b), which effectively cuts the downstream VMM input bus.

E. Mixed-Signal Processing Unit: Digital Buffer Block

The data to digital buffer blocks (DBB) in each quadrant are supplied from main memory via *Kp*–bit wide digital “load” bus (Fig. 1a). The DBB’s internal logic circuitry is designed to flexibly support various data flow scenarios - from simple load (Fig. 3b: scenario #1) to “load and shift” configuration (Fig. 3b: scenarios #2, 3). The flexible “load and shift” configuration enables efficient computation of CNV layers with a wide range of specifications (filter size and stride) while taking advantage of the row-wise data reuse (Fig. S1b).

Specifically, the input to each DBB is supplied by a *Kp*-bit-wide 2-to-1 multiplexer, which selects the input source between the load bus (i.e. main memory) and the previous DBB. For simplicity, we assume that only one DBB can be loaded from the load bus at one step, by setting the address of the target DBB on the buffer address bus. Similar to the data transfer from neurons, the number of processor cycles required for each step varies based on the location of the DBB.

The shift operations are supported by properly configuring a “shift-bit” flip-flop in each DBB and are masked by “shift enable” line. In particular, the shift register configurations are programmed by setting shift bits to “1” for all the DBBs except for the first block of shift register (Fig. 3b). Setting shift bit to “1” of a particular DBB is performed concurrently with loading that DBB from load bus by asserting “shift mode” control line. (Note that row-wise execution of the CNV layer typically starts by loading all DBBs with row data from main memory, with all shift operations disabled by de-asserting shift enable signal line, so that all shift bits are typically configured at this time.) Once shift bits are properly configured, simultaneously with loading new data into DBB, the already loaded data in DBBs can be shifted between the remaining blocks of that shift register (e.g. shifted to the right in scenario #2 and #3 of Fig. 3b). This is performed with internal “master shift command” signal, which enables latching data from previous DBBs of the specific shift register whenever its shift bit and enable signals are set to “1”.

Additionally, each DBB has a single-bit SR latch to specify the target VMM block columns, i.e. to set column enable (CE) signal, which is configured similarly to shift-bit flip-flop. It also has a bus repeater on the load bus, which is disabled when the DBB is loaded, to stop downstream data propagation on the bus (Fig. 3a).

F. Mixed-Signal Processing Unit: VMM Computation

Assuming that the target VMM columns are pre-selected (CE=1) and the desired row-select bits are set, VMM operation is performed by applying a positive pulse to “VMM_OP” command signal. This signal enables the neuron blocks associated with the selected rows as well as their input VMM blocks for which CE=1 via propagating through the RE and then CoE lines. Naturally, the VMM_OP pulse width should be longer than the worst-case end-to-end VMM operation time, i.e. the time it takes from the moment inputs are applied to DAC to that of latching ADC outputs.

G. Auxiliary Unit

Auxiliary unit (AUX) is provisioned to perform less frequent digital computations in neuromorphic inference. In particular, this block is used to perform in parallel *K p*-bit vector-by-vector operations such as additions, subtractions, and fixed-precision multiplications. It also performs comparison in max-pooling operation, which is typically used in CNNs. (An average pooling, another typical operation, can be implemented directly in a mixed-signal domain with properly adjusted weights.) AUX consists of an array of arithmetic logic units (ALUs), multipliers, and internal registers. During max-pooling operation, the register holds the current maximum value and feeds it to one of the ALU’s inputs to compare to the next input value fed from the main memory through the load bus. The outputs of AUX blocks can be written back to main memory via store bus.

H. Controller and Instruction Set Architecture

The most typical operations on aCortex involves loading multiple DBBs with data from the main memory, performing vector-by-matrix computation, and then moving the results from neuron blocks back into the main memory. Accordingly, aCortex controller includes three separate agents as loader, operator, and collector, each dedicated to performing one of these frequently-used operations (Fig. 4a). These agents are

TABLE I. MICROARCHITECTURAL PARAMETERS

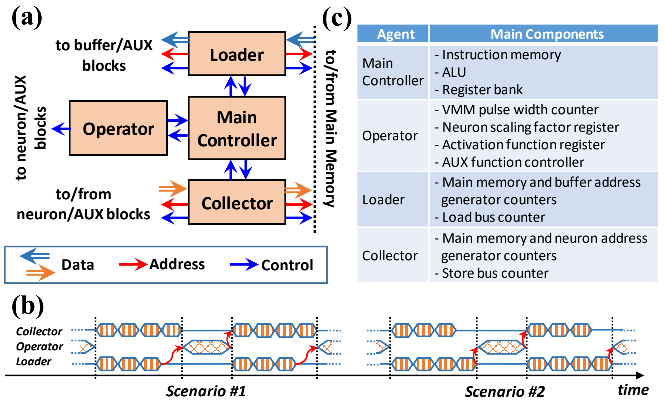
Parameter	Description	Typical
p	Input/weight/computing precision; also bit length of each data word	4
M	# digital buffer blocks / columns of VMM blocks in one MSPU quadrant	32
N	# neuron blocks / rows of VMM blocks in one MSPU	64
K	Computation granularity; also VMM block dimensions, buffer/neuron block size, and digital bus and main memory I/O width in terms of words	64

configured and synchronized using a main controller. Such multi-agent structure shrinks the code size, enables eligible time overlapping between these operations, and minimizes the engagement of the main controller, which in turn reduces the controller time overhead.

The main components of each agent are listed in Fig. 4c. The loader is responsible for reading data from the main memory into the DBB/AUX blocks using the load bus. This agent uses two counters to support burst mode in which multiple data packs with a pre-configured main memory/DBB initial addresses and strides are loaded. Moreover, the loader uses a load bus counter to adjust the load time based on the physical distance between the main memory and the target DBB. Such distance is extracted from the most significant bits of the DBB address considering that the location of neuron and DBBs are counted from the center of the chip outwards, starting from 1. For example, load_time = memory_read_time + ceiling(DBB_address/32) for the case when passing data on the load bus through 32 DBBs takes one controller cycle. The collector which is responsible for storing back the neuron/AUX's output to the main memory has a similar structure to the loader. The operator has an embedded counter to produce the appropriate pulse width of VMM_OP control signal for the VMM operation. It also holds the neuron scaling factor, the type of activation function selector, and AUX function control bits. The main controller instructions are further detailed in Section S.II.

III. APPLICATION MAPPING

Application mapping process of a given neural network on aCortex involves checking accelerator resource requirements (i.e. main memory and MSPU) and producing a weight mapping and machine code to setup the accelerator. Similar process is also used in the accelerator design to estimate the required main memory and MSPU resources, and to optimize their specifications (*M*, *N*, *K*) for the target range of applications. Such process starts with extracting the computational graph and finding the optimal topological partitioning and ordering of the computational steps (Fig. 5a). In such graph, the vertices are computation kernels (i.e. neural layers) while the edges are input/output data. In general, finding the optimal sorting of a given graph is an NP hard problem. However, layer-by-layer operation scheme of aCortex, resulted from our energy-optimal design, reduces the complexity of the problem. After obtaining the graph and scheduling order, the next step is to check the memory requirement throughout the inference steps. In aCortex, 3D data are stored in the main memory in a row first, column second, and channel/feature map third order (Fig. 5b). Following this order, multiple channels of one data pixel (and adjacent pixels in a row in the case of CNV stride >1) are always grouped into *K*-word-long data packs (Fig. S1b). One pack of *K*-word data is then mapped into a word line of main memory block and can be read/written simultaneously. Such data placement in the main memory enables burst mode read/write using controller's loader/collector. Note that due to quantization such scheme may result in underutilized memory, e.g. when the number of channels/feature maps is not divisible by *K*.



Agent	Main Components
Main Controller	- Instruction memory - ALU - Register bank
Operator	- VMM pulse width counter - Neuron scaling factor register - Activation function register - AUX function controller
Loader	- Main memory and buffer address generator counters - Load bus counter
Collector	- Main memory and neuron address generator counters - Store bus counter

Fig. 4. (a) Top-level representation of the multi-agent controller and connectivity between different agents and the rest of the architecture, (b) examples of agent operation timing diagram when the load operation is completed before (scenario #1) and after (scenario #2) the collect operation, and (c) the main components of each controller's agent.

Considering such data arrangement, the memory usage after each inference step (i.e. neural layer) is calculated by drawing a cut in the computational graph which separates the already computed portions of the graph (network processing steps) from the upcoming ones, and adding up all the edges that are crossed. Since during execution of each layer both its inputs and outputs are present in the main memory, the upper-bound for the total memory usage is calculated by adding up the memory usages for two consecutive cuts and subtracting their overlap edges.

We applied this algorithm to three studied networks, namely Inception-v1 [36] and ResNet-152 [37] DNNs for image classification, and GNMT [38], a Google's neural machine translation network featuring a 16-layer LSTM network with bi-directional encoder layers, with the vector length of 1024 and the sequence size of 10. For example, Fig. 5c shows such process for an Inception neural layer and an unfolded 2-layer LSTM network. As this figure shows, the memory requirement is limited by the initial layers of the DNNs (for 4-bit computation with the data pack quantization of $K = 64$).

In the next step the weight matrices are mapped into MSPUs using a greedy search algorithm (Fig. 6a), for which input parameters are the number of available MSPU's columns (*M'*), architecture granularity parameter (*K*), the number of tries (*epoch*), and the list of weight kernels (*LoK*). In one iteration of the algorithm, the kernels are first randomly ordered and then greedily mapped in a row-first manner, in the given order, to the array of VMM blocks. Such procedure is repeated *epoch* times and mapping configuration with the smallest number of occupied VMM blocks is selected. Furthermore, the mapping process is repeated for different $M' \leq M$ to search for a square shaped mapping of occupied VMM blocks (Fig. 6b) to minimize the average data transfer distance between MSPU's active VMM/neuron blocks and the main memory (and hence to increase energy-efficiency) – see more discussion on that in

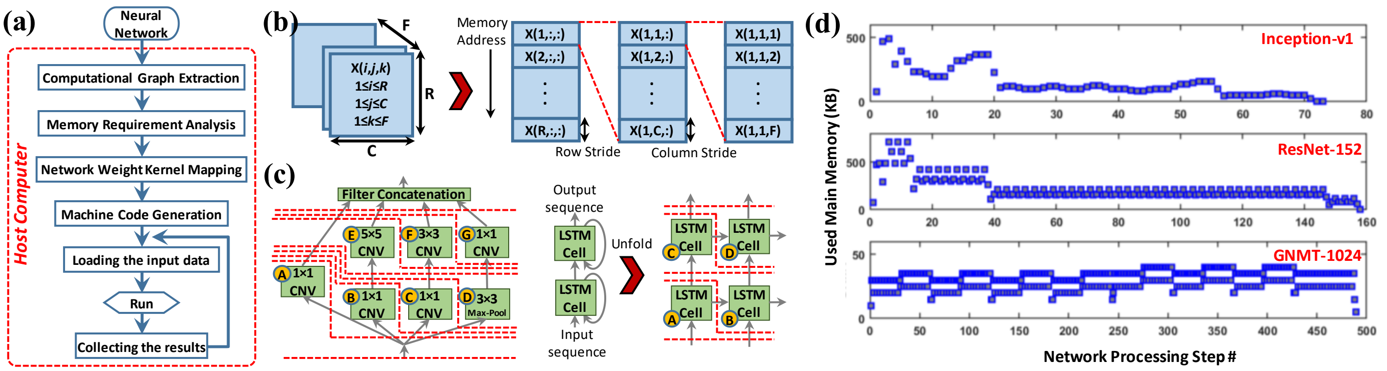
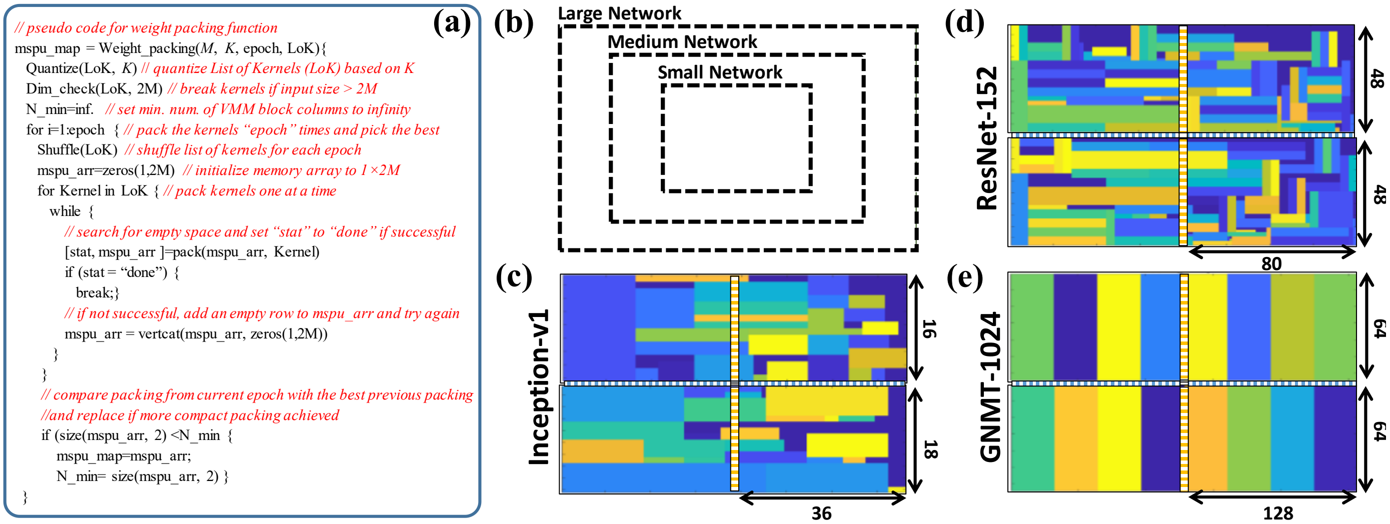


Fig. 5. (a) Application mapping flow-chart performed on the host computer. (b) Mapping scheme of a 3D data structure into the aCortex's main memory. (c) An example showing computational graph cuts for evaluating the amount of main memory occupied during various steps of inference for a single Inception layer (left) and a multi-layer LSTM network (right). (d) The utilized main memory graph as a function of network processing step for the studied benchmark networks.



```
// pseudo code for weight packing function
mspu_map = Weight_packing(M, K, epoch, LoK){
  Quantize(LoK, K) // quantize List of Kernels (LoK) based on K
  Dim_check(LoK, 2M) // break kernels if input size > 2M
  N_min=inf.   // set min. num. of VMM block columns to infinity
  for i=1:epoch  { // pack the kernels "epoch" times and pick the best
    Shuffle(LoK)  // shuffle list of kernels for each epoch
    mspu_arr=zeros(1,2M)  // initialize memory array to 1×2M
    for Kernel in  LoK { // pack kernels one at a time
      while  {
        // search for empty space and set "stat" to "done" if successful
        [stat, mspu_arr ]=pack(mspu_arr,  Kernel)
        if (stat = "done") {
          break;}
        // if not successful, add an empty row to mspu_arr and try again
        mspu_arr = vertcat(mspu_arr, zeros(1,2M))
      }
    }
    // compare packing from current epoch with the best previous packing
    //and replace if more compact packing achieved
    if (size(mspu_arr,  2) <N_min {
      mspu_map=mspu_arr;
      N_min= size(mspu_arr,  2) }
}
```

Fig. 6. (a) Pseudo code for aCortex weight kernel packing algorithm. (b) Preferred mapping locations of various neural network sizes. (c-e) Weight kernel mapping results for Inception-v1, ResNet-152, and GNMT-1024 for K=64. Each pixel shows one VMM block colored according to the neural layer occupying it.

Section IV-B. The results of this optimization process for our target networks are shown in Fig. 6c-e.

The mapped locations of weight kernels are then used in the compilation process to generate the machine code for handling the data flow and operation of the network. The number of instructions for initializing and operating each layer type is estimated for the system-level analysis based on the proposed ISA (presented in Sections II-H and S.II).

IV. Circuit Design and Performance Evaluation

A. 55-nm FG-based Implementation of Mixed-Signal Blocks

While aCortex can be realized with variety of NVM technologies, the focus of this paper is on a mature industrial-grade flash-memory technology that have already enabled extremely compact and energy-efficient implementations of mixed-signal circuits [20]. The key advantage of such implementation, as compared to, e.g., those based on resistive switching devices [31], is the FG cell's inherent signal amplification and low operation currents, which greatly relax the requirement for sensing circuitry gain, and enables very compact peripheral circuits [20]. Moreover, the experimental results for the chip-to-chip statistics, long-term drift, and temperature sensitivity of the FG-based mixed-signal circuit prototypes strongly attest to suitability of this memory technology for developing complex, practically-useful neural accelerators [17, 22]. More specifically, the proposed architecture was evaluated for the implementation based on ESF-3 (embedded split-gate flash) technology in which FG cells were redesigned for analog computing applications [17].

The proposed mixed-signal circuits allow for efficient implementation of different types of activation functions, kernel sizes, and quantization ranges, which are essential features of multi-purpose inference accelerator (Fig. 7a). In particular, the VMM operation is implemented using gate-coupled design [17]. In such approach, inputs are encoded as currents, which are applied to the peripheral FG transistors (Fig. 7a). The weights are encoded in the memory state of the array FG devices, while peripheral FG device are typically programmed to the low threshold states (maximum current) for optimal operation [17]. The multiplication of an input with the corresponding weight, is performed by a pair of FG transistors, one at the periphery and one inside the array. The currents from different array FG devices are summed up on the drain (row) line at the local sense amplifier to complete dot-product operation computation.

The front-end input conversion is realized using current steering (CS) DAC architecture, a viable choice considering its low power consumption, compact footprint, and fast turn on/off time at relatively low precision. A 4-bit PMOS-based CS DAC circuits source the current into peripheral FG devices.

The developed current-mode global sensing circuit of a neuron block has excellent wideband current following behavior and provides very low input impedance, while limiting drain voltage distortion. To reduce process variation overhead associated with an offset error, two additional FG devices are provisioned in each channel (drain line) and are used either to source or sink the input referred offset current.

The input current scaling of a neuron block is implemented using a binary-weighted current mirror structure controlled by the multi-bit digital input (Fig. 7a). As already mentioned in Section II.D, this feature is needed to adjust quantization range in accordance with the maximum VMM circuit output currents (which would vary, e.g., with the size of the weight kernels), and hence to minimize losses in functional performance due to quantization of activation function outputs.

The 4-bit current-mode ADC has a 1-bit per stage cyclic design which generates the 4-bit digital output in 4 cycles. FG transistors are also employed for offset/compensation in the high-speed comparator and to generate reference currents.

The sigmoid and ReLU activation functions are implemented directly in the ADC, without using any other additional circuitry, by appropriately choosing reference currents, i.e. quantization levels of the ADC. Linear and hyperbolic tangent functions are emulated via input biasing and appropriate weight scaling (Fig. 7b). DACs/ADCs are designed for unipolar data and can be reused, without any circuit modification, for bipolar data by utilizing offset-binary representation. More details on the circuit structure for the design of PVT-resilient CS DAC circuit and algorithmic ADC can be found in [20].

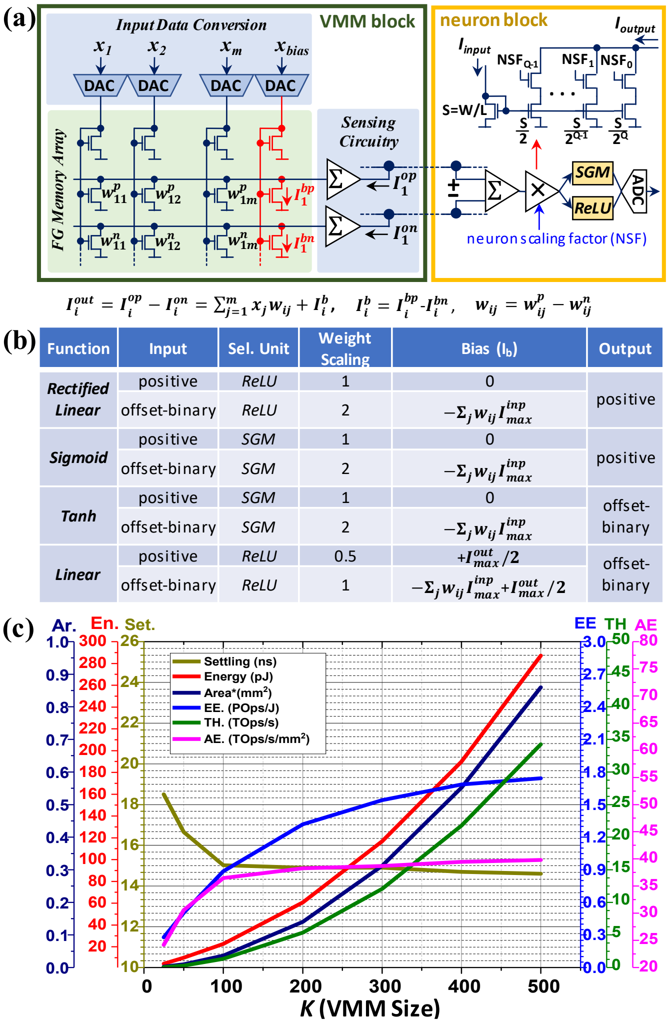
Fig. 7c provides the VMM performance results as a function of K based on the measurements of ESF3 memory devices and post-layout simulations of peripheral circuitry. These data are used to estimate system-level performance.

B. Main Memory and Buses in 55-nm Technology Node

The main memory is implemented using asymmetric 2T gain embedded DRAM cells with boosted power supply [39]. The retention time of eDRAM cells was changed to 100 µs by reducing leakage and adjusting biases, with 99.9% bit yield confirmed by block-level Monte Carlo simulations [39]. The memory performance was modeled using CACTI tool [40]. We also developed a bus area/energy/delay model as a function of bus length and repeater size using post-layout simulations considering all device/interconnect parasitics.

C. System-Level Results and Design Space Exploration

We have developed a software framework that utilizes the post-layout energy/speed/area metrics for all the aCortex's building blocks (buffers, buses, DACs, ADCs, integrators, and digital circuits) to evaluate the system-level performance for any target DNN/RNN network. This framework uses the list of processing tasks for a given neural network to map the VMM kernels on the NVM devices embedded in the VMM blocks, and then generate a detailed performance report for the given set of architecture specifications. Using such tool, we have performed a preliminary exploration of architectural parameters (i.e. K and MSPU aspect ratio, AR = M/N) to optimize the processor performance for the aforementioned target neural networks. A detailed study of these benchmark networks (Figs. 5, 6) has shown that a 1MB MM is sufficient to store all intermediate data, while the flow control program requires at most 4KB of instruction memory. Moreover, the controller energy/delay is estimated in an instruction-by-instruction manner in which the required machine code for initializing and performing each layer type has been evaluated.



$$I^{out}_i = I^{op}_i - I^{on}_i = \sum_{j=1}^{m} x_j w_{ij} + I^b_i, \quad I^b_i = I^{bp}_i\text{-}I^{bn}_i, \quad w_{ij} = w^p_{ij} - w^n_{ij}$$

Function	Input	Sel. Unit	Weight Scaling	Bias (I_b)	Output
Rectified Linear	positive	*ReLU*	1	0	positive
	offset-binary	*ReLU*	2	$-\Sigma_j w_{ij} I^{inp}_{max}$	
Sigmoid	positive	*SGM*	1	0	positive
	offset-binary	*SGM*	2	$-\Sigma_j w_{ij} I^{inp}_{max}$	
Tanh	positive	*SGM*	1	0	offset-binary
	offset-binary	*SGM*	2	$-\Sigma_j w_{ij} I^{inp}_{max}$	
Linear	positive	*ReLU*	0.5	$+I^{out}_{max}/2$	offset-binary
	offset-binary	*ReLU*	1	$-\Sigma_j w_{ij} I^{inp}_{max} + I^{out}_{max}/2$	

Fig. 7. (a) VMM/neuron schematic for the FG-based mixed-signal universal neural computing scheme supporting both positive/negative input/output as well as various (non)linear activation functions. (b) The configuration of the computing elements for various scenarios (desired activation function and input type). (c) Performance results for a VMM connected to a neuron block as a function of K assuming 4-bit computing precision, maximum cell current of 16 nA, maximum current of 1 µA and unity gain for local sensing, maximum ADC input current of 5 µA, and ADC input scaling range of Q = 5. Such numbers are selected based on optimal operation conditions [17] and our analysis of the weight and output distributions for the considered neural networks. Also, note that the area numbers are calculated for active circuitry only (i.e. no programming/erasure circuitry included).

Fig. S2 shows the system-level energy efficiency (EE), throughput, and area with respect to K and AR for these networks. Larger K typically results in higher throughput due to wider bus widths and consequently higher data transfer rate. It also improves the EE by reducing the VMM block peripheral circuitry energy consumption (trend clearly seen in Fig. S2c). Moreover, the increase in throughput results in lower leakage

TABLE II. ACORTEX SYSTEM-LEVEL RESULTS AND BREAKDOWNS *

	Inc.-v1	ResNet-152	GNMT
Network Specification			
Number of Parameters	7.2×10^{6}	5.52×10^{7}	1.3×10^{8}
Number of Operations	5.2×10^{9}	2.0×10^{10}	2.6×10^{9}
Architectural Specification			
K	64	64	64
M	38	80	128
***N* (top/bottom)**	16/18	48/48	64/64
Main Memory Capacity	1 MB		
Number of Memory R/W	3.3×10^{5}	8.1×10^{5}	1.7×10^{4}
Main Memory Utilizat. (%)	47.8	59.8	5.07
VMM block Utilization (%)	67.5	87.6	100
Area Breakdown (%)			
Main Memory	17.3	4.4	2.2
Sensing Circuitry	15.6	23.5	25.1
FG Arrays	24.5	36.8	39.3
DACs	4.5	6.8	7.3
Neuron Blocks	0.06	0.04	0.03
Programming/Erasure**	26.5	14.2	11.3
Others	11.5	14.2	14.8
Energy Breakdown (%)			
Main Memory	36.3	23.2	10.9
Sensing Circuitry	16.4	12	33.6
FG Arrays	3.5	2.5	7.1
DACs	6.8	4.9	13.8
Neuron Blocks	0.8	1.1	0.9
Buses	30.1	41.4	16.2
Leakage	3.3	13.5	16.8
Others	2.8	1.4	0.7
Performance Summary			
Area (mm^2)	37	146	293
Power (mW)	20.6	26.5	44.5
Inference Time (ms)	2.37	6.72	0.16
EE (TOp/J)	97	106.2	335.8
Throughput (TOp/s)	2.00	2.9	14.94

* Because of more optimal overlapped execution and refined estimates for DAC circuits, the numbers are slightly adjusted compared to preliminary ones reported in Ref. [19]. ** The area overhead of programming/erasure circuitry is estimated as $(245.5M + 120N)\times K$ μm^2.

energy which further increases EE. However, for the networks with medium and small weight kernels (i.e. ResNet-152 and Inception-v1), larger K results in under-utilization of active blocks, and buses, as well as an increase in the number of required VMM blocks to map the network, which in turn increases the energy overhead of the analog peripheral circuitry and buses. As such negative effects outweigh the positive ones, the overall system-level EE and throughput are getting worse for larger K. For DNNs, the load bus delay/energy typically plays a more significant role in the system throughput/EE compared to the store bus due to higher input-to-output data transfer ratio in convolution operations. Such property leads to higher throughput/EE for smaller AR (i.e. relatively shorter load bus) in these networks. The opposite of this trend is observed for GNMT in which the LSTM layers have smaller input-to-output data transfer ratio. Note that such trends do not consistently hold due to network-specific weight packing efficiency with respect to AR.

The accelerator area decreases for larger values of K due to larger sharing factor of analog peripheries, hence smaller area overhead of DACs and sensing circuitry (Fig. S2). This trend does not hold for Inception-v1 in which smaller weight kernels result in the VMM block under-utilization and lower weight packing efficiency which outweigh the gain in peripheral circuitry area efficiency.

The results for different networks indicate $K = 64$ and AR ≈ 2 as a semi-optimal design point for which block utilization and load/store bus energy/delay are somewhat balanced. Detailed performance report and area/energy breakdowns for this design point are presented in Table II. As these results show, the energy consumption is dominated by data transfer and intermediate data storage for DNNs with smaller size weight kernels, such as Inception-v1 and ResNet-152 networks. On the other hand, the energy consumption is dominated by sensing circuitry and DACs, even despite larger accelerator area (and hence larger data transfer energy consumption) for GNMT inference task. This is because GNMT inference involves larger size VMM operations (weight kernels), which ultimately leads to larger compute-to-communication ratio and allows to take better advantage of analog-domain computing. Also, note that area overhead of high-voltage programming/erasure and VMM peripheral circuits is quite low (as compared to other NVM-based accelerators) due to their effective sharing, and, in fact, the area of aCortex is dominated by FG memory cells. Thus, the detailed results show that the integration density is one of the key properties of memory devices for the energy-efficient inference accelerators.

The performance comparison of aCortex against its major fully digital [5-7] and mixed signal [23, 24] competitors shows that aCortex achieves a significantly higher performance, especially for mobile/IoT applications, for which the storage efficiency (MB/mm^2), and EE are the most important metrics (Fig. S3). In order to make a fair comparison, we performed a highly optimistic projection of the performance metrics for the mixed-signal architectures to 55-nm, 4-bit design point. According to these estimations, aCortex achieves ~28×/~65× improvement in EE over ISAAC [23] / PUMA [24], while maintaining a comparable SE and enduring a relatively small drop in throughput (~0.3×/~0.4×). Note that these architectures do not consider the overhead of programming/erasure circuitry which could impact the performance results.

V. CONCLUSION

This paper discusses aCortex, a novel multi-purpose mixed-signal architecture for accelerating neuromorphic inference. The presented architecture is optimized for energy efficiency, which is achieved by performing most of the computations and some of the data transfer in the analog domain and by maximizing sharing of peripheral and programming / erasure circuitry among VMM blocks. Simulation results for 4-bit aCortex implemented with embedded NOR flash memory in 55 nm process show record-breaking energy efficiency at favorable area efficiency as compared to previously suggested mixed-signal accelerators.

Though latency of aCortex is already sufficient for practical applications, its speed and computing throughput can be further significantly improved by performing more VMM computations in parallel, overlapping data transfer over the busses and VMM computations, and pipelining and/or increasing the bandwidth of the buses. Such modifications may require increasing the bandwidth of main memory and designing controller/ISA capable of handling multiple layer operations, and will naturally reduce the energy efficiency. Understanding energy efficiency – latency tradeoffs of aCortex architecture is one of the important future research goals.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature,* vol. 521, pp. 436-444, May 2015.

[2] V. Sze, Y.H. Chen, T.J. Yang, and J. Emer, “Efficient processing of deep neural networks: A tutorial and survey”, *Proceedings of IEEE*, vol. 105 (12), pp. 2295-2329, 2017.

[3] N.P. Jouppi, C. Young, N. Patil, and D. Patterson, “A domain-specific architecture for deep neural networks”, *Communications of ACM*, vol. 61 (9), pp. 50-59, 2018.

[4] NVIDIA Corp. Investor day presentation (2017).

[5] N.P. Jouppi *et al*., “In-datacenter performance analysis of a tensor processing unit,” in: *Proc. International Symposium on Computer Architecture (ISCA’17)*, Toronto, Canada, June 2017, pp. 1-12.

[6] Y. Chen *et al*., “DaDianNao: A machine-learning supercomputer,” in: *Proc. International Symposium on Microarchitecture (MICRO’14)*, Cambridge, MA, Oct. 2014, pp. 609-622.

[7] J. Lee *et al*., “UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision,” *IEEE Journal of Solid-State Circuits*, vol. 54, no.1, pp.173-185, 2018.

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

[9] C. Mead, *Analog VLSI and Neural Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[10] R. Sarpeshkar, “Analog versus digital: Extrapolating from electronics to neurobiology,” *Neural Computation*, vol. 10, pp. 1601–1638, 1998.

[11] S. Chakrabartty and G. Cauwenberghs, “Sub-microwatt analog VLSI trainable pattern classifier,” *IEEE Journal of Solid-State Circuits* , vol. 42, pp. 1169–1179, 2007.

[12] G. Indiveri *et al*., “Neuromorphic silicon neuron circuits,” *Frontiers in Neuroscience,* vol. 5, art. 73, 2011.

[13] J. Hasler and H. B. Marr, “Finding a roadmap to achieve large neuromorphic hardware systems,” *Frontiers in Neuroscience*, vol. 7, art. 118, 2013.

[14] F. Merrikh Bayat *et al.*, “Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits”, *Nature Communications*, vol. 9, art. 2331, 2018.

[15] M. J. Marinella *et al*.. “Multiscale co-design analysis of energy, latency, area, and accuracy of a ReRAM analog neural training accelerator”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, pp. 86-101, 2018.

[16] C. Li *et al.*, “Analogue signal and image processing with large memristor crossbars”, *Nature Electronics*, vol. 1, art. 52, 2018.

[17] X. Guo *et al.*, “Temperature-insensitive analog vector-by-matrix multiplier based on 55 nm NOR flash memory cells”, in: *Proc. Custom Integrated Circuit Conference (CICC’17)*, Austin, TX, Apr.-May 2017, pp. 1-4.

[18] M. Bavandpour, M.R. Mahmoodi, and D. Strukov, “Energy-efficient time-domain vector-by-matrix multiplier for neurocomputing and beyond,” in: *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

[19] M. Bavandpour *et al.*, “Mixed-signal neuromorphic inference accelerators: Recent results and future prospects,” in: *Proc. International Electron Devices Meeting (IEDM’18)*, San Francisco, CA, Dec. 2018, pp. 20.4.1-20.4.4

[20] M.R. Mahmoodi, and D.B. Strukov, “An ultra low energy internally analog, externally digital vector-matrix multiplier circuit based on NOR flash memory technology,” in: *Proc. Design Automation Conference (DAC’18)*, San Francisco, CA, June 2018, art. 22.

[21] G. W. Burr *et al*., “Experimental demonstration and tolerancing of a large-scale neural network using phase-change memory as the synaptic weight element”, *IEEE Transactions on Electron Devices*, vol. 62, pp. 3498-3507, 2015.

[22] X. Guo *et al.*, “Fast, energy-efficient, robust, and reproducible mixed signal neuromorphic classifier based on embedded NOR flash memory technology”, in: *Proc. International Electron Devices Meeting (IEDM’17)*, San Francisco, CA, Dec. 2017, pp. 6.5.1-6.5.4.

[23] A. Shafiee et al., “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in: *Proc. International Symposium on Computer Architecture (ISCA’16),* Seoul, Korea, June 2016, pp. 14-26.

[24] A. Ankit *et al*., “PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference,” in: *Proc. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’19)*, Providence, RI, Apr. 2019, pp. 715-731.

[25] X. Liu *et al.*, “RENO: A high-efficient reconfigurable neuromorphic computing accelerator design,” in: *Proc. Design Automation Conference (DAC’15)*, San Francisco, CA, June 2015, pp. 1-6.

[26] X. Liu *et al.,* “Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63 (5), pp. 617-628, 2016.

[27] L. Song, X. Qian, H. Li, and Y. Chen, “PipeLayer: A pipelined ReRAM-based accelerator for deep learning,” in: *Proc. Symposium on High Performance Computer Architecture (HPCA’17),* Austin, TX, Feb. 2017, pp. 541-552.

[28] P. Chi *et al.*, “PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,” in: *Proc. International Symposium on Computer Architecture (ISCA’16),* Seoul, Korea, June 2016, pp. 27-39.

[29] M. Imani *et al*., “RAPIDNN: In-memory deep neural network acceleration framework,” arXiv:1806.05794, 2018.

[30] H.Y. Chang *et al*., “AI hardware acceleration with analog memory: Micro-architectures for low energy at high speed,” *IBM Journal of Research and Development,* vol. 63 (6), pp. 8:1-8:14, 2019.

[31] D. Strukov and H. Kohlstedt, “Resistive switching phenomena in thin films: Materials, devices, and applications,” *MRS Bulletin*, vol. 37, pp. 108-114, 2012.

[32] F. Merrikh Bayat et. al, “Model-based high-precision tuning of NOR flash memory cells for analog computing applications”, in: *Proc. Device Research Conference (DRC’16)*, Newark, DE, June 2016, pp. 1-2.

[33] F. Merrikh Bayat *et al*., “Redesigning commercial floating-gate memory for analog computing applications”, in: *Proc. International Symposium on Circuits and Systems (ISCAS’15)*, Lisbon, Portugal, May 2015, pp. 1921-1924.

[34] A. Mishra, E. Nurvitadhi, J.J. Cook, and D. Marr. “WRPN: Wide reduced-precision networks.” arXiv:1709.01134, 2017.

[35] I. Hubara *et al*., “Quantized neural networks: Training neural networks with low precision weights and activations,” arXiv:1609.07061, Sep. 2016.

[36] C. Szegedy *et al*., “Going deeper with convolutions,” in: *Proc. Conference on Computer Vision and Pattern Recognition (CVPR’15),* Boston, MA, June-July 2015, pp. 1-9.

[37] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in: *Proc. Conference on Computer Vision and Pattern Recognition (CVPR’16)*, Las Vegas, NV, June-July 2016, pp. 770-778.

[38] Y. Wu *et al*., “Google's neural machine translation system: Bridging the gap between human and machine translation,” arXiv:1609.08144, 2016.

[39] K.C. Chun, P. Jain, T.-H. Kim, and C.H. Kim, “A 667MHZ logic-compatible embedded DRAM featuring an asymmetric 2T gain cell for high speed on-die caches,” *IEEE Journal of Solid-State Circuits*, vol. 4 (2), pp. 547-559, 2012.

[40] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi, “CACTI 6.0: A tool to understand large caches,” Technical Report, HP Labs, HPL-2009-85, 2009.

Supplementary Information

S.I. Neural Layers and Weight-Stationary Hardware Compatibility

Artificial neural networks (ANNs) consist of a common core computing "neuron" cell. Capturing the basic behavior of its biological counterpart, the artificial neuron calculates the weighted summation of inputs passing through a (non)linear activation function, *f*(), as $y = f(\sum_{i=1}^{m} x_i w_i)$ where *x* and *w* are inputs and weights, respectively. Targeting various applications, different multi-layer ANNs with various layer-types (i.e. neuron connectivity and activation functions) have been developed [1]. We next briefly review the layer operation for today's most popular ANN models, which, e.g., occupy 95% of Google's data center workload [5], and present their weight-stationary hardware-friendly re-arrangement.

A. Fully-Connected (FC) Layer

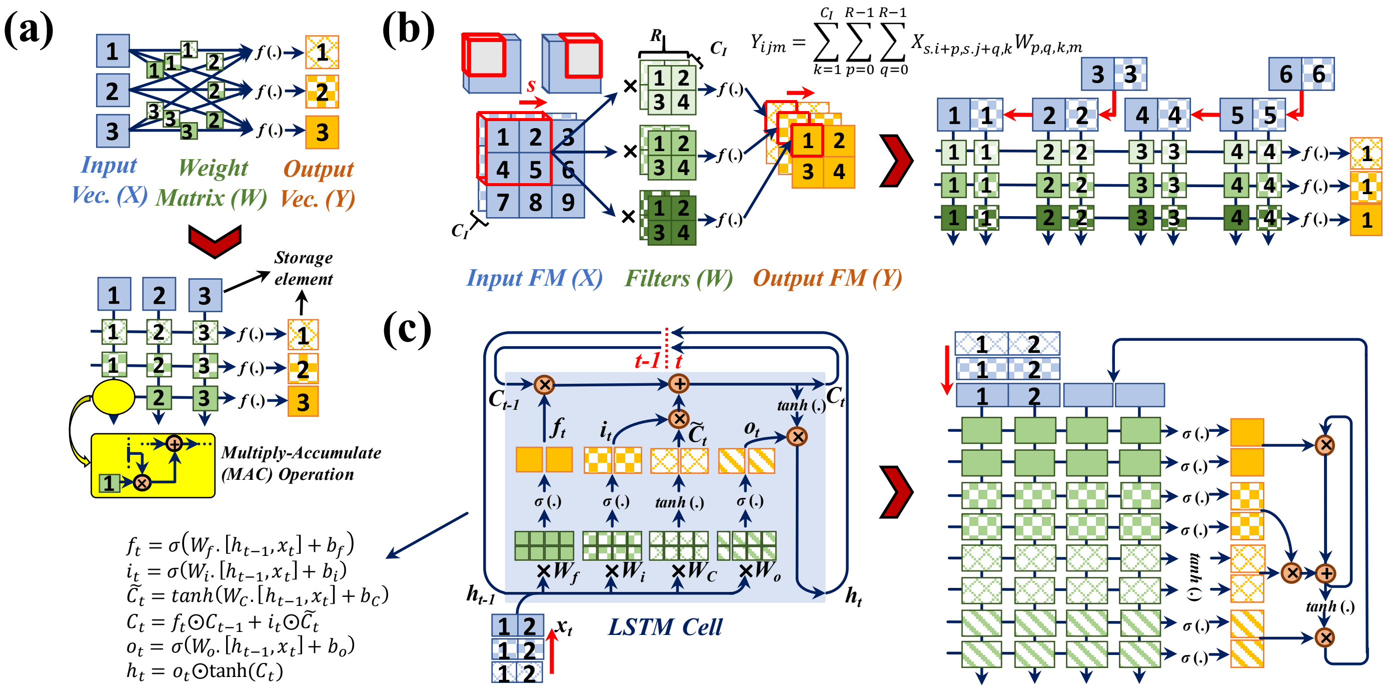
FC is the most common ANN layer, e.g. in multi-layer perceptron (MLP) networks, which are used for classification, prediction, etc., and convolution neural networks (CNN), which are mainly used for image classification/recognition [1]. In the FC layer, each neuron in the input layer feeds all the output neurons through a set of weights. As shown in Fig. S1a, FC layer can be re-organized as a weight-stationary VMM followed by an activation function. In such VMM, weight matrix (green) is stationary, and the input elements (blue) are vertically shared and propagated through all the weight columns at the same time. Accordingly, all weight locations simultaneously perform multiply-and-add operation, and the outputs (shown with orange color) are calculated in parallel.

B. Convolutional (CNV) Layer

CNV is the core neural layer of CNNs, the dominant network in computer vision, which uses a special connectivity pattern to efficiently exploit the spatial locality of inputs while extracting image features. This layer type includes multiple channels of 3D weight matrices, each applied over the whole 3D input data in a sliding window fashion, to produce its corresponding output feature map as shown in Fig. S1b. This figure also shows our target scheme to map the CNV operation into a weight-stationary VMM structure. In this scheme, CNV operation is performed in a row-first manner in which one output pixel (for all the channels) is calculated at a time. At the input, the CNV row-wise data reuse is exploited using multiple "load and shift" chains of input buffers. This input buffer array feeds a stationary 2D weight matrix, which is a reshaped and stacked representation of the 3D CNV filters. Accordingly, the output vector represents different channels of one output pixel location at a time. Note that in this scheme, different input channels of each input pixel location (C_I elements) can be grouped in one input data pack without disturbing the data flow (Fig. S1b). Moreover, for the strides larger than unity ($s > 1$), every *s* adjacent pixels in the row direction can also be grouped into one input data pack. Hence, for a given CNV operation, the data pack size can be any divisor of $C_I \times s$.

C. Recurrent Layer

Recurrent neural layers aim to extract and interpret the information encoded into the temporal locality in a sequence of inputs using a feedback connection and a sequential operation. Long-short-term-memory (LSTM) is one of the most popular recurrent layers, which is widely used in language translation and speech recognition [1]. Fig. S1c shows the original LSTM structure and its weight-stationary re-arrangement. As shown,


(a)
Input Vec. (X)
Weight Matrix (W)
Output Vec. (Y)
f(.)
Storage element
Multiply-Accumulate (MAC) Operation
(b)
s
R
C_I
Y_ijm = Σ_{k=1}^{C_I} Σ_{p=0}^{R−1} Σ_{q=0}^{R−1} X_{s.i+p,s.j+q,k} W_{p,q,k,m}
Input FM (X)
Filters (W)
Output FM (Y)
(c)
t-1 t
C_t-1
C_t
f_t
i_t
C̃_t
o_t
tanh (.)
σ (.)
×W_f
×W_i
×W_C
×W_o
h_t-1
h_t
x_t
LSTM Cell
f_t = σ(W_f.[h_{t−1}, x_t] + b_f)
i_t = σ(W_i.[h_{t−1}, x_t] + b_i)
C̃_t = tanh(W_C.[h_{t−1}, x_t] + b_C)
C_t = f_t⊙C_{t−1} + i_t⊙C̃_t
o_t = σ(W_o.[h_{t−1}, x_t] + b_o)
h_t = o_t⊙tanh(C_t)

Supplementary Information Fig. S1. Network structure and hardware-friendly representation of the most popular neural layers: (a) fully-connected layer, (b) convolution layer, and (c) long-short-term-memory (LSTM) layer, all targeting a weight-stationary dataflow scheme. Note that the red arrows represent the dataflow, and the numbers represent location index.

the LSTM's computational effort is dominated by VMM operations for which the input vector is obtained via concatenating the current element of the input sequence and a hidden state [h_{t-1}, x_t]. Accordingly, the VMM's weight matrix is obtained by stacking the forget (f), candidate (C), input (i), and output (o) weight matrices. The rest of the LSTM computation includes basic element-wise vector operations and recurrent data transfer for the next step. Note that unlike CNV, the LSTM computation includes bipolar inputs and multiple activation function types which calls for a more generalized computing scheme supporting such cases – see Fig. 7b and its discussion in Section IV.A. Moreover, LSTM layers typically have very large weight matrices resulting in a larger compute-to-communication ratio. Hence the computing efficiency typically plays a more significant role (compared to data transfer efficiency) in the overall efficiency of inference task for such layers.

S.II. Instruction Set Architecture Details

The main controller instructions are:

Agent configuration: CNF *agent*, (*mstr*), (*nstr/bstr*)

Configure the agent (i.e. loader/collector) with appropriate parameters such as memory stride (*mstr*) and neuron/buffer stride (*nstr/bstr*) which are immediate fields in the instruction.

Load: M2B *rm, rb, cnt, smode, sen*

Command the loader to load *cnt* data packs (each *K* words) starting from initial memory address specified by register *rm* into the digital buffer blocks starting from the initial address specified in register *rb* (assuming that strides are pre-configured). *smode* field specifies the new value loaded to the shift bit in each digital buffer block, and *sen* enables "load and shift" operation.

Compute: VMM *nsf, af*

Command the operator to start VMM computation while simultaneously configuring the neuron scaling factor (*nsf*) and activation function type (*af*).

Collect: N2M *rn*, *rm, cnt*

Command the collector to collect *cnt* data packs (each *K* words) from neuron blocks starting from initial address specified by register *rn* into the memory locations starting from initial address specified in register *rm* (assuming that strides are pre-configured).

Row select: RSEL *n_addr*

Set the row select bit in the neuron block specified by *n_addr* to "1".

Synchronize: WAIT *agent*

Hault the main controller until the target *agent* finishes its task.

Reset: RST

Reset all the column and row select bits.

The remaining instructions include simple arithmetic (i.e. *add/sub*, *addi/subi*) and (non-)conditional control (i.e. *jmp*, *djnz, call, return*) instructions. Note that for the considered applications, all data in main memory are used before they have to be refreshed. Therefore, for simplicity, we will not discuss refresh operation, though its implementation is straightforward and can be performed explicitly using either M2B instruction or automatically by memory controller.

S.III Prior Work

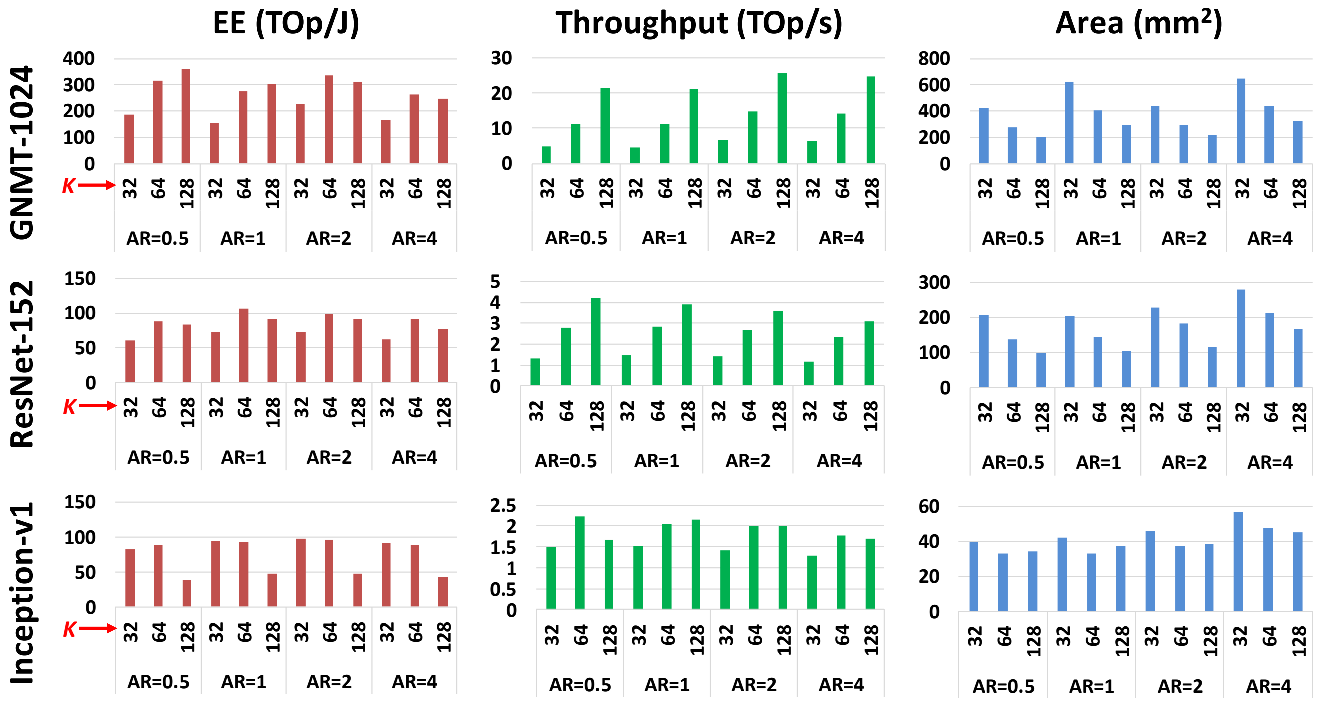
At the system-level, many efforts have been recently made to exploit the efficiency of mixed-signal operators and develop an efficient DNN/RNN processor architecture [23-30].

Specifically, ISAAC [23] and PUMA [24] architectures are 2D mesh structure of tiles where each tile contains several small fixed-size ReRAM-based VMM units (typically 128×128) with dedicated input/output peripheral circuitry. In these architectures, one shared memory is implemented in each tile for storing intermediate data and communication between VMMs, while the communication between the tiles are performed through a shared 2D bus structure. Such heavily granular multi-core design approach is followed with the aim of increasing the utilization, minimizing the data transfer overhead, and maximizing throughput via pipelining and parallel processing. However, data conversion and communication overhead due to partial VMM operation, static power consumption of the analog blocks, large area overhead of the neurons / DACs / ADCs, and large control and communication overhead between tiles/VMMs limit the performance of such architectures, especially when running relatively complex computational graphs such as of Inception and ResNet.

RENO [25] and Harmonica [26] are, respectively, a ReRAM-based reconfigurable neuromorphic computing accelerator and a heterogeneous computing system based on such accelerator. This accelerator utilizes a mixed-signal centralized mesh interconnect network to reduce DAC/ADC overhead while increasing the throughput via passing the analog output directly to the next layer. However, this approach is only optimized for fully-connected multi-layer networks and associative memories. Moreover, despite lowering the DAC/ADC overhead, the accelerator performance is impacted by costly mixed-signal routers.

Pipelayer [27] explores the trade-off between hardware resource of ReRAM array and performance utilizing the notion of parallelism granularity targeting both training and inference. This architecture uses a spike-based integrate and fire scheme to eliminate DAC/ADC overhead. However, input multi-level encoding and output spike generation overhead still results in inferior efficiency.

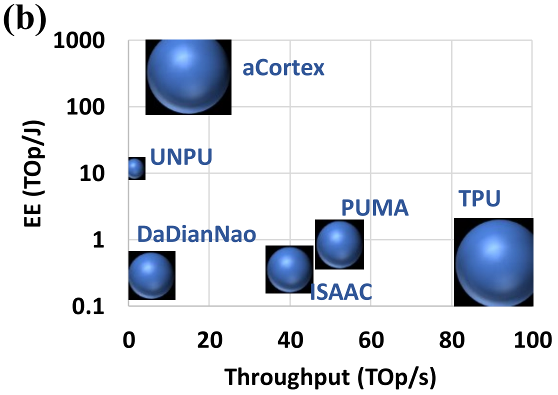
PRIME [28] is a ReRAM based architecture proposing the application of morphable memory blocks with small extra add-on circuitry which can be configured as computational unit on demand. Such morphable memory blocks result in a compact and energy efficient design by reusing the memory block peripheries for computation. However, the performance of PRIME is negatively impacted by lack of data-reuse for convolution, high data conversion/transfer overhead due to small analog-domain VMM, i.e. kernel breakdown, and latency overhead due to SA/neuron sharing. Additionally, the very limited switching endurance of ReRAM makes the main idea of PRIME hardly practical.



Supplementary Information Fig. S2. Design space exploration for aCortex performance metrics, i.e. energy efficiency (TOp/J), throughput (TOp/s), and area (mm^2), with respect to the key architectural parameters, i.e. granularity (K) and aspect ratio, (AR=M/N) for three benchmark neural networks (GNMT-1024 RNN; ResNet-152 and Inception-v1 DNNs).

(a)

	DaDianNao [6]	TPU [5]	UNPU[7]#	ISAAC [23]	PUMA [24]	aCortex
Technology node	28 nm	28 nm	65 nm	32 nm	32 nm	**55 nm**
Approach	digital	digital	digital	ReRAM	ReRAM	**2D-NOR**
Clock (MHz)	606	700	200	1200	1000	**700**
Precision (bits)	16 fixed point	8 fixed point	1-16 (4 here)	16 fixed point	16 fixed point	**4 fixed point**
Area (mm^2)	88	330	16	85.4	90.6	**293**
Power (W)	20.1	40	297	65.8	62.5	**0.044**
Throughput (TOp/s)	5.54	92	1.38	39.9	52.31	**14.94**
SE (MB/mm^2)	0.2	off-chip	off-chip	0.74 (0.25*)	0.76 (0.257*)	**0.273**
EE (TOp/J)	0.286	0.43	11.6	0.35 (5.14*)	0.84 (12.09*)	**335.8**



Supplementary Information Fig. S3. (a) Performance comparison of aCortex with the state-of-the-art digital and mixed-signal neuromorphic processor architectures. Except for TPU, all performance results are based on simulations. * Highly optimistic mapping of performance metrics to 4-bit computing precision and 55-nm technology node. #The performance numbers do not include overhead of external memory access (weights/intermediate data). (b) Energy efficiency versus throughput scatter plot for the approaches listed in (a). The size of bubbles represents the area of the processors.

RAPIDNN [29] is also a ReRAM based architecture which aims to improve the hardware performance through minimizing the required computing precision while achieving similar network accuracy. Precision is reduced by utilizing a reinterpretation mechanism (non-linear quantization of inputs/weights/outputs based on statistical data). Moreover, in this architecture all neural functionalities are implemented inside the memory using a direct digital lookup table-based technique which eliminates costly DAC/ADC/neuron. However, semi-sequential VMM operation and lack of data-reuse for convolution result in performance drop for large scale neuromorphic applications especially those involving convolution operation. Besides, the architecture suffers from data encoding overhead despite eliminating data conversion overhead.