CLASSIC: a Cortex-Inspired Hardware Accelerator

V. Puente, J.A. Gregorio
University of Cantabria
Santander, Spain
vpuente@unican.es

***Abstract*—**

This work explores the feasibility of specialized hardware implementing the Cortical Learning Algorithm (CLA) in order to fully exploit its inherent advantages. This algorithm, which is inspired by the current understanding of the mammalian neo-cortex, is the basis of the Hierarchical Temporal Memory (HTM). In contrast to other machine learning (ML) approaches, the structure is not application dependent and relies on fully unsupervised continuous learning. We hypothesize that a hardware implementation will be able not only to extend the existing practical uses of these ideas to broader scenarios but also to exploit CLA's hardware-friendly characteristics.

The architecture proposed will enable the system size to be scaled up compared to a state-of-the-art CLA software implementation. It may be possible to improve performance by 4 orders of magnitude and energy efficiency by up to 8 orders of magnitude.

Given the problem's complex nature, we found that the most demanding issue, from a scalability standpoint, is the massive degree of connectivity required. We propose to use a packet-switched network to tackle this. The paper addresses the fundamental issues of such an approach, proposing solutions to achieve a scalable proposal. We will analyze cost and performance when using well-known architectural techniques and tools. The results obtained suggest that even with CMOS technology, under constrained cost, it might be possible to implement a large-scale system. We found that the proposed solutions enable a saving of ~90% of the original communication costs running either synthetic or realistic workloads.

Keywords – Cortex, cortical columns, mini-columns, neurons, packet-switched network, neuroscience, computer architecture

1 Introduction

Mammal brains have a distinct structure compared to other biological systems: the presence of the neo-cortex. The salient feature of this construction is that it is anatomically and functionally remarkably homogeneous. Eighty years ago, Lorente de Nó [1] discovered that the neo-cortex (from now on, cortex) is composed of cylinder-like packs of a few thousand neurons forming *columns*. Later, V. Mountcastle [2] anatomically detailed the structure of these columns, which is approximately two millimeter high, 300-500 μm wide structures, where in most mammals a set of six layers can be distinguished (Figure 1.a). The columns are connected via low-range axons to other nearby columns (via Layer I and Layer VI) or other distant columns and the thalamus (via Layers V and VI). The columns are composed of ~30 μm wide structures called mini-columns (See Figure 1.b) [1]. Regardless of the functionality of each region, the cortex is quite regular, for example, in humans it has a surface of ~1600 square centimeters with few anatomical differences throughout it [2]. This fact has puzzled neuroscientists for decades: how such regularity can be the underlying structure for the complex functional organization of the cortex [3].

In neuroscience the most accepted hypothesis [1] is that the cortex is some sort of memory system. Its inputs are composed of incoming signals from the senses and the outputs are the actions on lower level brain structures. These are then sent to the motor system and expressed as behavior. The main hypothesis is that the cortex is continuously building a model of the world according to the information flow. This model is used to produce behavior.

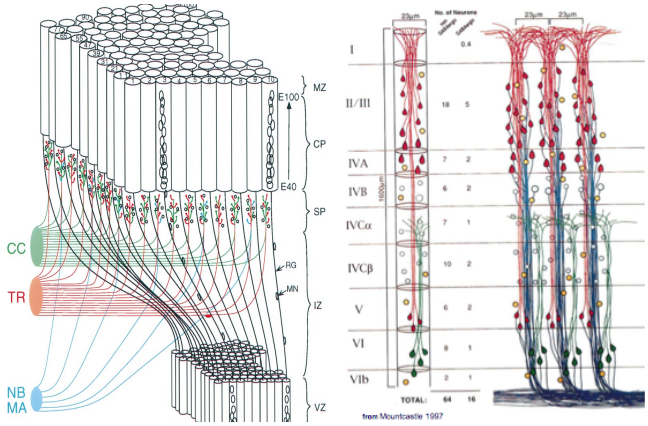


Figure 1 (a) Cortical Columns (b) Mini-columns

Initially George and Hawkins [6] hypothesized how the cortex might implement such a memory structure in order to fulfill the biological requirements. A theory has been built on the hypothesis that the cortex is a prediction device, that works as a self-associative memory and is hierarchically structured as a Hierarchical Temporal Memory (HTM) [7][8]. The theory, which is primarily influenced by current neuroscientific knowledge, includes an algorithm, called the Cortical Learning Algorithm (CLA) that provides the rules for storing and retrieving information, i.e. learning and making predictions. The idea has been used in practical problems such as anomaly detection, sequence prediction, pattern identification, natural language processing, etc. Currently HTM only considers the supra-granular layers of a cortical column. The modeling of the infra-granular layers in the cortical column [9] is under study and once this task is done, the hierarchy's inner workings can be considered. Therefore, CLA only contemplates intra-cortical column communication.

Given the current state, other deep-learning techniques (highly specialized, and in most cases using off-line learning), such as [10], produce better results in constricted problems. In contrast with these approaches, CLA has two remarkable properties: the core algorithm does not change from application to application,

and, like in biological systems, it is continuously learning. Coincidentally, both properties are required to achieve Artificial General Intelligence (AGI), as defined by AI theorists [11]. Even in the emerging state, CLA has already proved its advantages over other state-of-the-art techniques, in anomaly detection [12], continuous unsupervised learning [13][14], natural language processing [15], etc.,

Currently the progress made with CLA is based on software. There are many implementations, NuPIC [16] is the most remarkable one and it is supported by Numenta using open source licensing (AGPLv3). Other companies, such as IBM [17], are working using their own implementations. Although the practical uses increase the complexity of these tools, the core algorithms are simple. This approach provides the flexibility necessary to explore new practical uses or new core algorithm variations. Nevertheless, the software limits the system size to a few thousand mini-columns, which might restrict the practical uses or advancements in hierarchy definition. To circumvent this problem, it seems necessary to develop feasible hardware [17]. While proposals such as [18][19] advocate the use of HTM-only implementations, others, such as [20][21] propose the use general purpose architectures designed to be used with many machine learning algorithms, HTM being one example of use [22][23]. Within the conventional ML realm, there are many specialized and limitedly flexible hardware implementations, [24][25][26].

While some might argue that it could be too early to cast a specialized silicon product, given the algorithm and application development status, perhaps we need to start addressing the issues we might encounter later. In contrast with other machine learning (ML) approaches, such as Deep Neural Networks (DNN), CLA might bring complicated challenges. Instead of precisely weighted connections and computing intensive matrix multiplications, CLA's foundation is a hyper-connected, complex and highly dynamic topology to store and retrieve information. From a naïve hardware perspective, this is hard to achieve (a single mini-column can potentially be connected to thousands of different mini-columns). Although emerging technologies, such as 3D stacking and Non-volatile memory might ease these stringent requirements, a feasible implementation in a conventional CMOS process would be advantageous from the point of view of scalability [24].

Furthermore, CLA has a relevant advantage over DNN or other weight-based learning: it requires only very low precision (around 4 bits might suffice) to add and compare instructions. Unfortunately, using conventional architectures, software implementations cannot capitalize on this since topology handling requires frequent and costly data movement throughout the memory hierarchy, which might complicate the use of GPGPU programming models. A hardware implementation could greatly improve both the energy requirements and performance. It seems feasible to process many millions of samples per second within a constrained energy envelope. However, DNN, even using a significantly more costly multi GPU configuration, is far from achieving that performance [27].

This paper explores this path, presenting the architecture of a feasible hardware implementation. In contrast with [19], we will use architectural methodologies/techniques similar to those used in commercially available products, such as general purpose chip multiprocessors. Based on the biological properties of axons and dendrites, we define a system that uses a logical construct to fulfill the topological flexibility of CLA over an on-chip network. Given the low computational requirements, by attaching some simple logic to the routers of this network and with some memory to store the connectivity status, it could be possible to implement CLA without requiring complex and power-hungry general-purpose CPUs or GPUs.

Consequently, like in biological systems, the network is the point towards which the system gravitates. We will focus our attention on the communication substrate and procedures to make CLA feasible. We will describe how, using a packet-switched network and diverse computer architecture techniques, we can achieve a practical implementation. Different solutions will be presented to guarantee system scalability. We will analyze, through detailed simulation and using well-known modeling tools, the temporal and energy requirements of the system. The set of proposals introduced minimizes communication overheads. The combination of all these techniques on average reduces network delay and active energy requirements by ~90%. Since communication seems to be the most demanding issue, we believe that it might be feasible to construct a highly scalable hardware-based accelerator.

2 Background and Motivation

Before getting into the proposal details, first we will provide a brief introduction to the main concepts used by CLA and explain how software limitations might make a hardware-specialized implementation of the algorithm attractive. Although the interested readers might wish to consult the details in the bibliography provided, we hope that they can obtain an understanding of the core components, which are surprisingly simple and elegant.

2.1 Sparse Distributed Representation (SDR)

Empirical evidence [4][5] suggests that the neural system represents information using *sparse* activity patterns. In this representation [6], in contrast to conventional binary data representation (also known as *localist* [7]), each bit has semantic meaning. In this way, the data representation is highly resilient to a noisy and faulty environment (as is the biological one). Therefore, changing a few bits in the representation always produces a value with "similar" meaning to the original. To convert a localist representation (which can be any multidimensional data representation), an encoder has to be used. The encoder will expand the original data by thousands of bits where, at a given time, only a few can be *set* (typically ~2%). Note that, besides resilience, an inherent property of SDR is the low power requirements. For example, with a perfect encoder, a 2048-bit SDR value will require up to ~280 bits in a localist binary representation (i.e. $log_2 \binom{2048}{40}$). Therefore, the SDR will require, on average, 3.5x fewer bit activations (140 average bit flips vs 40). Another salient property of SDR is the Union property (to store multiple data in the same representation with a low probability of false positive identification) [8], behaving

as a space-efficient probabilistic data structure that resembles some of the properties of Bloom filters [9]. Strikingly, the basic principle is simple: the number of combinations of a few elements in a large set is so large[1] that a low number of coincidences enable the identification of a value with a very low probability of error. Perhaps biology, through evolution, has achieved a similar design. The foundation of CLA is that the cortex uses SDR-like representations to store and retrieve information.

2.2 Hierarchical Temporal Memory and Cortical Learning Algorithm

Currently CLA focuses on partially replicating the functionality of the cortical columns. Layer I is mainly used for interconnecting near cortical columns. Layers II/III, usually denoted as *inference layer*, are supposedly devoted to predicting the state of the cortical column in the next input steps. Layer IV, denoted *sensory layer* [1], handles the input signals to the column coming from the thalamus. Layers V and VI, handle the output from the cortical column to sub-cortical brain regions (such as motor commands) and lower level columns in the hierarchy (such as feedback) respectively. Inference layer predictions are used to compose the column output, which is forwarded to higher level cortical columns in the hierarchy. The thalamus acts as a relay point for the inputs (via primary order relay nuclei) or other cortical columns in the cortex (via high order relay nuclei) [10]. To avoid storage redundancy, one of the most accepted hypotheses is that different cortical columns are connected hierarchically (mainly layer I/VI to layer IV of remote columns).

In any case, the organization of this hierarchy (i.e. how the layers in different regions interact) is not well understood in neuroscience ([11],[12]–[14]). Therefore, the connections to other regions are not actually considered by the CLA algorithm. Although there is an ongoing research effort to support it in HTM [15] (which coincidentally diverges from the classical view of hierarchy), we have focused our attention on a single cortical column. Even in the current state, the CLA algorithm is enough, from a practical standpoint, to produce a useful system. The reader should note that the purpose of the theory is not to mimic brain functionality (at least, currently) but just to use it as inspiration to implement an alternative prediction system for time series.

The CLA defines the term mini-column, which is sufficient to handle hierarchy-less prediction (see Figure 2). A *proximal*[2] *dendrite segment* [16] could be connected to a subset of the bits of the input (which will be provided by a localist-to-SDR encoder). This restriction models the fact that the input action potential (i.e., spikes) will be observable from a subset of the mini-columns. The segment models the dendritic growth of the feed-forward connection of the system. It is well known that dendritic plasticity is responsible for the learning in the cortex [17]. For a given input, the intersection between the active inputs

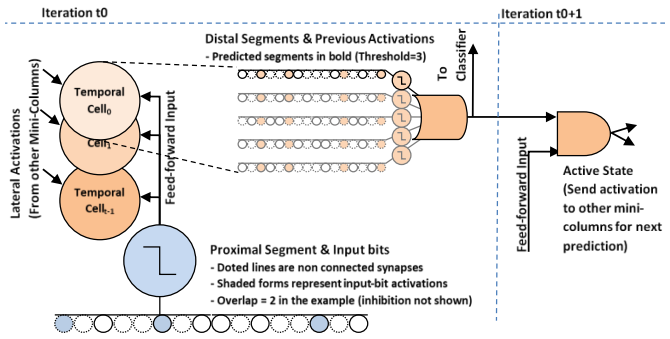


Figure 2 Graphical Summary of CLA algorithm

and connected synapses in the segment is determined in each proximal dendrite segment. The size of this intersection is called *input overlap*. When determined, an *inhibition process* will select the top ~2% of mini-columns with the largest input overlap. The remaining mini-columns are inhibited. The synapses to active inputs in the winning mini-columns are strengthened and synapses to inactive inputs weakened [18]. To handle learning, each synapse connection is tracked with a *permanence* value. If the value is above a predefined threshold, the synapse is considered connected. At boot time, the values are chosen randomly near the threshold value. In the CLA terminology, this is called *spatial pooling*. Therefore, the Spatial Pooler [19] is in charge of producing a stable SDR-compliant representation of each input value [20][21][22].

The Spatial Pooler output (i.e the inhibition winners' mini-columns) is forwarded to the component that should predict the next input according to the current state of each mini-column and the feed-forward input. The mini-column state is kept in Cells. Cells will establish relationships with other cells in other mini-columns. Such relations are tracked by synapses, grouped in *distal dendrite segments*[3] (see Figure 2). Building a sequence of these Cell-to-Cell segments, it is possible to predict which mini-columns will win the inhibition in the next input-sample.

To build this value, each cell can have three possible states: predicted, non-predicted and active. A cell will be *active* if its mini-column wins an inhibition while *predicted*. When the number of synapses connected to all active cells in the current cycle of a particular *distal dendrite segment* (i.e. the active cells overlap) is above certain threshold, the owner cell enters in the *predictive* state. A mini-column with a cell in the predictive state is expected to be an inhibition winner in the next iteration. Therefore, the set of mini-columns with at least one cell in the predictive state is the prediction of the cortical column.

In the next input-sample, all cells correctly predicted will perform the learning in the dendritic distal segments that produced the prediction. The strength of the synapses of the segment from cells active in the previous iteration is increased and decreased in synapses from non-active cells [18]. For the cells incorrectly predicted, i.e. in predicted state but the owner mini-column did not win the inhibition, all the active synapses

[1] For example, $\binom{2048}{40} \approx 10^{84}$, i.e. more than atoms in the Known Universe (~10^{76}-10^{82})

[2] Note that although the name comes from the term used for dendrites close to the soma (or cell nucleus) in pyramidal neurons (they are the most numerous excitatory neuron types in mammalian cortical structures), CLA does not model the neurons at low level.

[3] Although the algorithm used here follows [16], which only includes the basal dendrites, there is an effort to extend this by differentiating the role of the apical dendrites into developing a posterior temporal memory.

of the segment that provided the incorrect prediction will be decreased by a small amount.

If a mini-column is a winner of the inhibition but has no cells in the predictive state, all the cells in it burst. In one of them (called the learning cell) new synapses to a subset of the *previously* active cells will be added. This learning cell is chosen to be the owner of the distal segment with most synapses to the previously active distal cells. If the number of synapses is not enough (for example, the number is below the prediction threshold), randomly chosen new ones to other active cells will be added. If there are no distal segments to choose, a new distal segment will be created in the least loaded cell in the column. At boot time, there are no distal segments in the system. Non-learning bursting cells are used to grow the already present synapses in the next iteration.

In summary, with every new iteration, the 2% of mini-columns with the largest number of synapses in the proximal segment connected to the input are used to predict the 2% of mini-columns active in the next input sample. The next input will be used to build, reinforce or weaken the predictions.

Since the same mini-column/s can encode the same input in different sequences (or contexts), the cell prediction/activation is used to disambiguate the specific context. Therefore, multiple cells are required per mini-column. Nevertheless, even with a low number of cells, the number of "contexts" that the system can store for the same value is large. For example, in a system with 2048 mini-columns with 32 cells per mini-column, 40^{32} different temporal contexts can be represented for the same value.

The CLA terminology used for this task is *memory sequencing* and it is done in the Temporal Memory [16].

2.3 Encoding and Classification

To provide an SDR representation, in a practical scenario, an encoder is needed. There are a few rules that an encoder should obey in order to fulfill the SDR properties [8]. For a scalar encoder:

- The SDR representation of similar scalars should have a high number of set bits in common. Overlap should decrease smoothly as scalars become less similar.
- The SDR representation of dissimilar scalars should have very low overlap.
- The SDR representation for a scalar must not change during the lifetime of the system.

These conditions are fulfilled using a really simple approach (e.g. by constraining the range of values that can be represented) or a rather complex one (e.g. with large memory requirements and/or large encoding costs).

To better understand how SDR works, next we will describe a simple, yet hardware-feasible encoding strategy. Let's assume we need to encode a positive integer, L, into a N-bit SDR representation S with w active bits.

We use $seed_1=L \ div \ w$ and $seed_2=1+L \ div$ as seeds of a pseudo-random generator. Let *{Low}* be the group composed of the initial w unique elements generated by the first seed (where the operation modulo $N-1$ has been applied*)*. The *{High}* group is generated by the second seed (with values not present in *{Low}).* We choose the last $w-L \ mod \ w$ indexes of *{Low}*. The remaining $L \ mod \ w$ bits to be set in S are chosen in order from *{High}* group. A pseudocode of this encoder is presented in Figure 3.

Note that for any L, there is n such that $n \cdot w \leq L < (n+1) \cdot w$. The idea is to generate a unique set of random numbers smaller than $N-1$ for each interval. The SDR codification is obtained by using a sliding window that chooses for each L, a group of bits to set from its current interval and the remaining bits from the next one. For example, if $L=nw$, the SDR encoding of L is the w randomly generated numbers with seed $L \ div \ w$. If $L=nw+1$, $w-1$ bits are chosen from the same set and 1 bit from the set generated in the next interval (i.e., the one generated with seed $1+ L \ div \ w$). In this way, close scalars will have a high number of bit-sets in common. The probability of having two close representations for two separate integers is negligible for a sufficiently large N.

```
t_SDR encode(uint L) :
        randomGen.seed( L div w)
        t_SDR  low
        while low.length() != w:
           int col = randomGen.random() mod (N – 1)
           if col is not in low:
              low.append(col)

        randomGen.seed( 1 + L div w)
        t_SDR  high;
        while high.length() != L mod w :
           int col = randomGen.random() mod (N – 1)
           if col is not in low:
              high.append(col)

        t_SDR S = merge(low  [ (L mod w) - 1 : w - 1   ] ,
                        high [      0        : L mod w ] )
        return S
```

Figure 3 (a) HW-Friendly SDR Encoder

Since pseudo-random generators are deterministic, this approach does not require storing the conversion, only the logic necessary to perform the operations. Since other non-scalar time series can be remapped to scalars, we can consider that the encoding problem is not a relevant issue from the point of view of this study.

In contrast with encoding, classifiers are application dependent components. For example, detecting anomalies in a signal is straightforward but predicting multiple steps in the future can be very memory intensive. An Anomaly Detector Classifier is just a component that computes the fraction of unpredicted mini-columns (just bit masking current input SDR with the last prediction). Nevertheless, a Value Predictor Classifier needs a complex memory structure to store and lookup the current SDR prediction to determine the next "localist" input. Like in [24], an optimal way to achieve the desired flexibility is to run the classification problem in a general-purpose core.

2.4 Software Limitations and Hardware Opportunities

ANN renaissance in the form of Deep-Learning [23][24] has been motivated by the large raw computational power of state-of-the-art heterogeneous multi-GPU/multi-CPU systems. This has enabled the use of a consolidated theory in increasingly challenging problems, jumping from simple pattern recognition of handwriting [25], to enabling a machine to win in a complex game against the best human [26]. The algorithms underlying

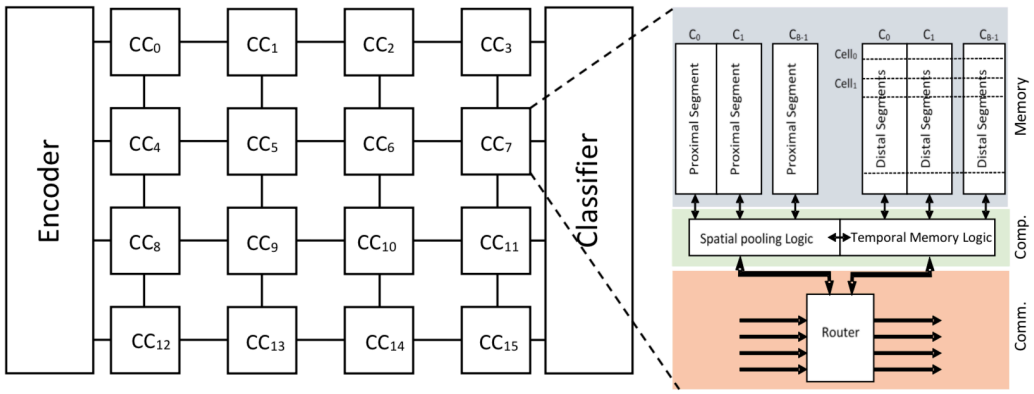


Figure 4 (a) The Columnar Cortex, (b) High-level description of a Columnar Core (CC)

these problems are suitable for data level parallelism, where GPGPU models excel [27].

In contrast, CLA's inherent nature makes it difficult to exploit such a paradigm. The synapses, although they require much simpler computations, can change dynamically. This difficult data level parallelism extraction will make GPGPU quite inefficient. Currently the support for this computing model in NuPIC is not even initiated. Perhaps, as happened with DNN in the past, CLA might not be able to take full advantage of the current and forthcoming hardware advancement, which might constrain the potential of the idea.

A CLA custom hardware accelerator will not only overcome these limitations by breaking the performance/energy barriers imposed by general purpose CPUs but will also take advantage of CLA's simple computations and low storage requirements. An insight into this advantage is that the core of most Machine Learning approaches is floating-point (matrix) multiplication and CLA only requires low precision integer addition/comparison. For example, according to [28], 32b FP multiplications will require 100 times more energy than to add two 8b integers. Similarly, being able to use only on-chip memory will reduce the memory access energy by two orders of magnitude.

Today ASIC-based DNN, such as [29][30][31][32], is appealing if the data and precision required fits the resources and the algorithm is well defined. In CLA this might be quite different because of its distinctive properties (i.e., there is no problem-specific customization, low memory requirements, low precision computing). Then, a hardware implementation of CLA may be more general purpose than a DNN one.

A CLA based ASIC might be useful in future applications. For example, it might be feasible to use thousands of streams of data concurrently and, using unsupervised learning, to detect anomalies. It is not easy to forecast the potential openings, but if we are able to perform fast CLA Natural Language Processing (NLP), such as [15], it could be possible to tackle challenging problems.

Finally, to really explore the full potential of the hierarchical organization and propose and validate theories about the underlying and unknown working mechanisms of the cortex, a hardware implementation would be useful. Under these circumstances, it is interesting to explore the feasibility of a silicon-based implementation, as this paper does. Next, we will introduce the architectural details of a potential implementation that we have called CLAASIC.

3 About the Feasability of CLAASIC

CLA's basic assumption is that synaptic plasticity is the key element used by the cortex to learn (through dendritic growth [34] as a consequence of the back propagation of post-synaptic action potential [17]). The relation between mini-columns is used to store and retrieve information. Such relation is defined dynamically depending on the connections established via on-line learning. Therefore, the storage capacity is proportional to the product of the number of mini-columns by the maximum number of connections per mini-column. The connectivity of the neurons can potentially be very high (the dendritic spines can provide up to tens of thousands of potential synapses). Nevertheless, most of these synapses are not active (i.e., the pre-synaptic axon is too distant from the dendrite) or multiple active synapses correspond to the same pair of neurons (hypothetically as a redundancy mechanism). Instead of electrically replicating the morphology of biological systems, which perhaps is unattainable, we will embed this functionality in a packet-switched network. We will focus our interest on how to organize and optimize the communication substrate to emulate axon spikes and correctly apply the prediction and learning algorithms of CLA. Instead of using synapses to establish a connection between two mini-columns, we will use memory structures attached to each router, modeling dendritic segments and the required logic performing the spatial pooling and providing temporal memory. Figure 4 (a) presents a high-level description of the proposed architecture. The encoder and the classifier will play the role of I/O interface with the general-purpose processor. We will implement the actual CLA mechanics in a component called the Columnar Core (CC). In this example, we will use a sixteen-core system connected by a packet-switched mesh network. Figure 4 (b) shows a high-level representation of a CC. In this case, we will assume that each CC is homogeneous. Next, we will briefly discuss the requirements of each component to later focus our attention on the most relevant one: the communication substrate.

3.1 Communication Requirements

The interconnection network must handle all the traffic generated by the CLA algorithm. The traffic has four classes: (1) input traffic incoming through the Encoder, (2) inhibition traffic,

(3) lateral activity due to cell activations and (4) mini-column activation and predictions sent to the classifier. This activity will be done at logical level using packets instead of physical wires. For example, each output bit of the Encoder will be connected to a statically defined set of columns (denoted *receptive field*). Then, for a given input, each active bit in the SDR representation will be transformed into a multicast packet, addressed to the potentially connected mini-columns. The Encoder will need a table with the relation between columns and inputs. Therefore, a multicast packet will emulate each axon spike. Similarly, when a mini-column enters in the predictive state, a unicast packet will be sent to the classifier.

Internally the router will receive messages from the spatial pooling logic (input overlap used in inhibition) and the temporal memory logic (cell activation). Those messages should be sent to the potential receptors. To maximize system performance, all mini-columns can be potential receptors. For example, for global inhibition, any mini-column should be aware of the input overlap of the rest of the mini-columns. The overlap is computed as the count of connected synapses in the proximal segment of the mini-column for a given input. With this information, the pooling logic can determine whether the current mini-column is among the 2% with highest overlap and can feed-forward the temporal memory logic. Similarly, to construct distal segments, the algorithm assumes that each mini-column is aware of all the cells in the active state. Consequently, any outgoing message from the spatial pooler or temporal memory will be broadcast to all the Columnar Cores in the system.

At first sight, the communication requirements seem demanding. There is a large amount of multicast/broadcast traffic that will require broad network bandwidth and large energy consumption. Additionally, any of the computations performed in the computing layer must be done accessing only local information. Since we cannot rely on any centralized component to scale the system to thousands of CCs, it is not evident how to achieve such synchronized behavior under realistic constraints.

3.2 Computing Requirements

There are two stages in the CLA algorithm that must be applied sequentially:

Spatial Pooler. The logic in charge will evaluate the input activity. The computing logic will evaluate the input overlap with its proximal segment (i.e. the number of synapses connected to an active input) and broadcast its value (assuming global inhibition) to the rest of columns in the system.

Inhibition logic might be quite straightforward (assuming global inhibition). Therefore, in each mini-column, we only need a counter to record the remote mini-columns with the largest overlap. If at a certain point this counter is greater than 2% (i.e. 40 for a 2048 mini-column system), the mini-column will self-inhibit for the current iteration, ignoring the remaining traffic. To break ties, additionally to the input overlap, each mini-column will include its ID in the inhibition packet. This *k-winners-take-all* inhibition is assisted by the network mechanisms (fundamentally, in-network replication and synchronized drain) to not only accelerate the task but also to convey and compare the incoming inhibition message overlap, counter increment (if overlap is greater than local), and comparison if the counter is above the desired sparsity.

The synapses in the proximal segment table of the active inputs will be adapted if the mini-column wins the inhibition. Therefore, the spatial logic will require a comparator, a 4-bit adder and a counter. The maximum overlap required is $\sim Log_2 Input$ bits. For a 2048 input encoder, 12 bits will suffice.

Note that although mini-column boosting might be required to achieve a balanced mini-column activation pattern [34], according to [20], it is not cost effective to do so in resource-constrained environments like an ASIC.

Temporal Memory. The module should handle feed-forward and cell activity. If we assume that the axon of the cells is global (i.e. we can form relationships between cells in any mini-column), a broadcast will be generated when a cell is active. The corresponding packets will include the origin cell. At destination, these will be kept in a list of current activations. Once the current iteration completes, the computing logic will determine for each mini-column whether the activation was correctly predicted. In this case, the corresponding distal segment of the cell in the predictive state will be updated accordingly (i.e. performing dendrite adaptation according to [18]). If the mini-column insn't correctly predicted, the logic should grow new synapses in the distal segment with higher overlap with the previously active cells (or create a new one if there aren't any). From a naive perspective, this can be difficult since it requires an extensive search through all the dendritic segments of the mini-column.

The temporal memory should determine for the current activations which dendritic segments in the mini-column are active. The *(temporal)* cells with an active dendritic segment will generate a broadcast/multicast in the network. Finally, the mini-columns that are not predicted correctly will produce a burst, which from the network perspective is equivalent to an activation.

3.3 Memory Requirements

The precision required by the algorithm is low. In a practical problem such as [35], there is no appreciable performance loss (<1%) when tracking the synapse permanence from full 64b FP precision to 4-bit integers. The reason for this is that there is low sensitivity to learning rates. In this application Temporal Memory uses steps of 0.1 (with permanence between 0 and 1). The spatial pooler uses smaller values (0.08 for learning and 0.003 for forgetting), which can be modeled with 8-16 levels stochastically controlled. This is biologically plausible, since in-cortex dendrite growing/shrinking is a stochastic process [33].

The proximal segments will store the permanence of the synapses with each potentially connected input bit. Note that each bit of the SDR representation produced by the encoder is potentially connected (i.e. a synapse might be formed) to the chosen subset of mini-columns at boot time. In general, we can assume that each bit can be connected to any mini-column in the system. Therefore, the proximal segment must have one entry for each potential input. In practice, each mini-column will be connected (i.e. a synapse will be formed) to a subset of encoder inputs. Thus, we might structure the proximal segment as a conventional cache indexed by the input index. In practice,

having capacity for 64-128 entries in a 2K mini-column system seems to be enough. The permanence value must be stored there. Reduced precision weights in DNN have a much more adverse effects on system performance [31][32]. For example, if we assume a 2K mini-column system with 1K inputs, the aggregation of all cortex proximal segments will require (including tags) between 0.25MB and 0.5MB. However, issues such as conflicts must be considered. The random nature of the SDR encoding implies low address locality.

In a naïve approach, each distal segment will require as many synapses as mini-columns in the system and each cell might have an unbounded number of segments. In practice, for a 2K mini-column system, having 128 segments with 40 synapses per cell provides similar results to an unbounded system [16]. Therefore, excluding tags, ~80KB will be required for 4-bit precision per mini-column, i.e. 160MB for the whole system. This amount seems feasible to achieve in on-chip SRAM memory storage such as is used here. Although orthogonal to this work, note that this is a raw amount that can be greatly reduced using the appropiate techniques. Exploiting CLA's noteworthy fault resilience (>50% rate of faulty cells can be tolerated [16]), it could be possible to reduce the final storage requirements significantly.

3.4 About the Temporal Cost of the Computing Phase

A key insight is that the learning (the most complex part of the algorithm) is outside the critical path. Since prediction only requires comparisons and counter increments, it is reasonable to assume that the time required to perform the prediction will be memory bounded. Since the memory required per mini-column is quite small, with the proper SRAM configuration, it could be possible to use a pipeline with a sustained throughput of one packet served per cycle.

The learning algorithm, especially distal segment formation, is more complex. Nevertheless, assuming a uniform distribution of mini-column activation, learning on one mini-column will be done with sparse frequency, i.e. only ~2% of the iterations. Therefore, in the best case, the time budget for the learning is 50 times larger than the prediction. Since (See section 4.3) computation can be fully overlapped with the communication phase, the time available for prediction is equal to the time available to perform the communication. Since learning in CLA resembles [18], forgetting is only performed on inhibition winner mini-columns. This is different from conventional STDP learning rules, which require the decrease of all non-winning synapses connected to the input [36].

Although the logic in charge of this is not analyzed in this work, it is reasonable to consider it is not relevant from the hardware perspective (both in time and area). In contrast with other works such as [37][38], fully optimization for CLA will not necessary require complex operation support. In §6.4, we will discuss how a non-uniform mini-column activation pattern might affect this using a realistic workload.

4 Communication in the Columnar Cortex

We have identified three major problems in the CC: communication and synchronization, temporal memory logic complexity and distal segment organization. From a scalability standpoint, the most relevant seems to be the former, since the necessary scalability appears to be a key element in the cortex. Next, we will discuss the key issues for the communication substrate and how we propose to deal with them.

4.1 Network Characteristics

Since all the spikes will be modeled as multicast packets, to optimize performance, the router requires multicast support (i.e, in-network replication). This feature can be included with minimal impact using a router similar to [39]. In-network replication will also reduce energy requirements, since the copying of the packet is performed through the path to destination. Finally, it will achieve a low base latency, since there is no injection serialization.

The packet size required is quite small. Inhibition traffic will require overlap and tie-breaker ID ($Log_2NumColumns + Log_2NumEncoderActiveInputs$). Lateral activity will require the source mini-column and cell ID ($Log_2NumColumns + Log_2NumTemporalCells$). Input activity will require source ID ($Log_2NumInputs$). For a 2048 column/input system, with 32 cells per mini-column, the size required will be 22, 16 and 11 bits respectively. Although, these sizes are much smaller than in a conventional CMP (where in most cases, the packets are around tens of bytes), the cortex organization or further enhancement (such as Section 4.5) might require adjusting the bandwidth availability (i.e., link width).

Since the individual latency of a packet is not critical, a low-degree network with narrow links might satisfy the requirements. High-degree networks would require increasing the complexity of the routers and the wiring cost. Therefore, 2-D Torus or Mesh [40] will meet these requirements. Although not explored here, like in biological systems, CLA gracefully tolerates faulty/noisy input or internal system degradation [16][41]. Therefore, it will also tolerate a faulty network. With a fault tolerant network such as [42], it could be possible to scale up the system size without yield issues even using wafer-to-wafer-on-wafer 3D integration under aggressive technological nodes, as [38] suggests.

4.2 Distributed Synchronization

Looking at the algorithm, there are four main phases: computing overlap of the proximal dendrite with the current encoded input, determining the winning mini-columns in the system, determining the lateral activity in each cell in the mini-column and producing the prediction. Overlapped with these phases, the adaptation (i.e. learning) of the synaptic segments is performed.

The main difficulty of performing such tasks in a fully distributed way is to know *when* each one should be performed. For example, input overlap should not be run until all the input activity is received (i.e., each mini-column has received all input packets). There will be no acknowledgment message of axon spike reception. Therefore, each CC should be aware when to run the corresponding part of the algorithm. Similarly, inhibition cannot be activated until each mini-column is aware whether it is one of the most active ones. Finally, prediction cannot be done until all active cells are known. The simplest, yet most efficient, way to circumvent this problem is to drain the network content

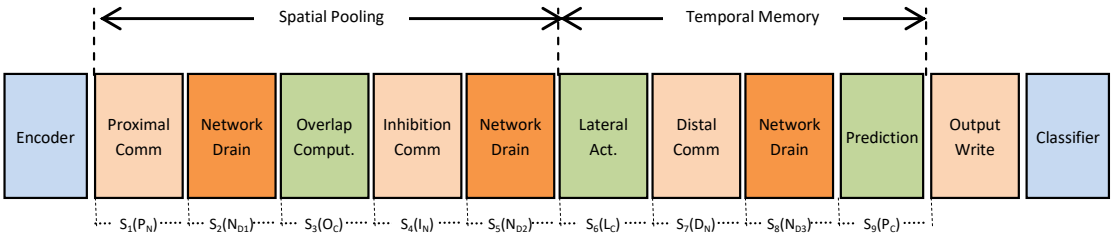


Figure 5 Stages in CLA Algorithm

before progressing to the next phase. If the network is empty, there is a guarantee that all the influencing packets will already have arrived at their destination.

Figure 5 details all the stages required for the CLA algorithm. Besides encoding and classifying, there are nine stages, three of them perform computation in the spatial and temporal logic (S3, S6 and S9), three correspond to the axon spikes (S1, S4 and S7) and another three are required to drain the network (S2, S5 and S8).

The problem of synchronization is then reduced to providing a scalable network drain mechanism. To guarantee the scalability of such a mechanism, we need a cost-efficient way to do so within the network itself. A simple approach is to use dimensional order routing (DOR) [40] and inject a special broadcast packet, denoted *broom packet*, into the extreme Columnar Cores from the smallest and largest ID (in the example in Figure 4, these should be CC_0 and CC_{15}). These packets will be allowed to progress to the next routers only if the local router has no more packets in the injection queue and the transit buffers at the ports where the router has received the copies of the packet are empty. The packet is replicated through the remaining ports. For example, when CC_5 receives the CC_0 *broom packet* from CC_4 and CC_1, we know that there are no normal spike packets that might affect the columns handled by CC_5. When the transit queues from W and N are empty, the router replicates the CC_0 broom packet through the S and E ports. This operation will be applied in the whole cortex until the columnar core CC_{15} receives the broom packet from CC_0. At this point, CC_{15} is aware that there are no packets in the network. Then, it can progress to the next stage in the algorithm. Similarly, when an intermediate CC receives all the *broom packets* from CC_0 and CC_{15}, it knows that there are no pending packets in the network for it. It should be remarked that this mechanism operates in a fully distributed way and will scale according to the network's available bandwidth.

We hypothesize that in biological systems, this drain is not required because the input rate of change is slow enough to guarantee that the spatial and temporal memory are handled satisfactorily. When the input rate is too high, the system will be unable to learn or predict. As a naïve example, an excessively fast image rate of change will be perceived by the visual cortex as noise. Although a similar solution can be applied in our case, we think that encoder and data are not evolutionarily tuned like in biological systems and perhaps will require an excessively long worst-case delay to work correctly in corner cases.

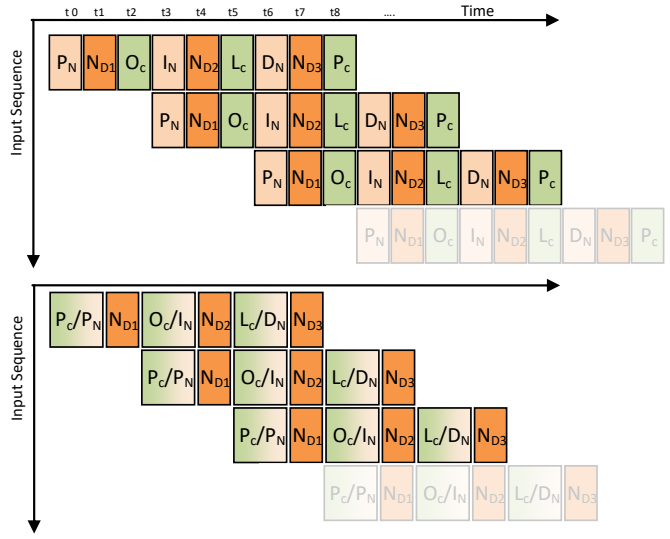


Figure 6 (a) CLA pipelined algorithm, (b) Overlapping communication and computation

4.3 Pipelined Algorithm: Communication and Computation Overlap

The nine stages in the algorithm will require a substantial amount of time and energy. Specifically, the network seems to play a fundamental role, since it is foreseeable that the time required to propagate the axon spikes will be large. Nevertheless, if we look at Figure 5, we can identify stages like in a general-purpose processor.

Therefore, we can use the same optimization techniques used there. We can pipeline the algorithm reducing the stages per input sample to three. Figure 6.(a) shows how that organization will be beneficial once the pipeline is loaded. The idea is to start computing the overlap of the next input in the sequence as soon as we know the current overlap. Then at *t3*[4], two operations are being performed simultaneously in the network. If we move forward in time, we can see how we can overlap three different input operations in a single stage. At *t6* the information interchange corresponding to the activation of cells for the first value, the inhibition for the second value and the input SDR of the third value are being processed by the network. At *t8* we are simultaneously performing the prediction for the first iteration, the lateral activity computation for the second one and the overlay computation for the last one. Even more importantly, we

[4] This could be a system-status dependent number of clock cycles depending on the computing logic and the network characteristics.

will need only one network drain per input iteration. Once the pipeline is loaded, we need only three iteration s in the input sequence to produce a prediction.

This approach creates the opportunity for further improvements. We do not need to finish the computation phases before starting to send the outcome of each one (i.e. we can fully overlap the computation and communication phases). As soon as spatial and temporal logic starts to generate axon spikes, they can be injected into the network, as Figure 6(b) shows. Therefore, the number of clock cycles required to process a value in the input sequence will be determined by the slowest portion: communication or computation. The number of cycles required by the slowest one and the clock cycle will determine the time required to process one sample in the input sequence. Finally, network drain should be synchronized across iterations: *broom packets* are forwarded in the CC router both if there are no packets in the injection queue and transit buffers and if all the local columns have finalized the current iteration (in the spatial and temporal logic). Therefore, network drain operates as a synchronization barrier.

4.4 Traffic Aggregation: Coalescing Injectors

To minimize communication cost, combining multiple mini-columns in a single CC is an effective solution. To use links between routers with a very short distance can unnecessarily increase the average latency in the network. To reduce this delay, the size of the CC (i.e. number of columns) should be tuned to match the propagation delay with the network clock cycle. This is well known for a Non-Uniform Cache architecture [43]. With this approach, it could be possible to aggregate action potentials coming from multiple mini-columns in the same CC in a single packet. Although this might increase the number of flits of the packet, it will reduce the network load.

Finally, the pipelined algorithm creates the opportunity for additional traffic aggregation. We can combine actions coming from different stages in the algorithm in a single packet. For example, inhibition can be combined with the lateral activations of the previous iteration. To do so, we assume the existence of coalescent injection queues (similar to the structure used to support non-blocking caches in general-purpose processors). Before queuing new packets in the injection buffer of the router, the packets waiting to be injected are checked. If there is a match in the destination mask, the previous packet is modified appending the information about the new one and then discarding it.

4.5 Scaling Traffic: Scale-out Zones

Biological systems would lead us to believe that the best approach to increase system storage is to increase the number of mini-columns and not the number of cells (and distal segments) per mini-column. From a practical perspective, if we increase the number of mini-columns, we might reduce the number of distal dendritic segments required per cell. Although from a software perspective this does not seem interesting, from the hardware standpoint, it is relevant because it might reduce the interconnection cost and perhaps the complexity of the CC. Therefore, in a hypothetical silicon implementation it would be desirable to increase the number of columns as much as the technology allows, i.e. depending on the yield and/or power envelope. Unfortunately, the communication system, as described at this point, might not scale up beyond a limited point.

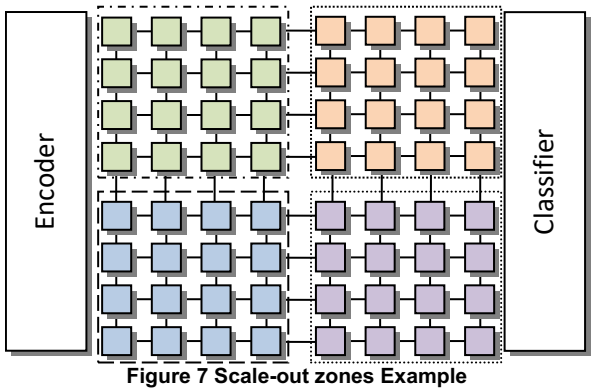


Figure 7 Scale-out zones Example

Distal and inhibition traffic are assumed to be global by the CLA (although inhibition might be local, this is rarely used because the performance falls significantly[5] and a loss of accuracy is incurred [20]). From the network perspective, the delay and power requirements will be increased significantly as we increase the number of CCs. Note, that the number of columns involved in the inhibition process is substantially higher than the number of inputs active in the encoder.

Biological systems do not use global communication in such processes. Inhibition, which is performed by inhibitory interneurons [44], should have a localized and static radius. Similarly, distal activity is constrained to the shape of distal dendrites and axons of pyramidal neurons [45]. Proximal traffic is less demanding because the CLA algorithm assumes that potential proximal synapses are limited. At boot time, each mini-column can potentially be connected to a subset of inputs, called the receptive field [46] of the column. Usually the receptive field of each mini-column is a subset of the input bits following a topological arrangement. This improves the accuracy of the system. Coincidentally, it reduces proximal traffic relevance, since the destinations in the multicast packet will be localized in the same region of the cortex.

To circumvent the global communication problem, we propose a simple approach that is based on splitting the network into separate zones and restricting the inhibition and distal traffic within them. We denote these regions as *scale-out zones*. For example, Figure 7 shows those zones used to increase the number of CCs from 16 to 64. Instead of requiring broadcasts, the traffic generated by columns in any of these zones will be restricted to them. If we need to further increase the number of mini-columns, we can simply increase the number of zones. With this simple approach, traffic will be kept constant.

The encoder, i.e. proximal traffic, selects the potentially connected columns without making distinctions between zones,

[5] Due to requiring the computation of the average distance between all connected synapses and their respective mini-columns in the input and the SP to update the inhibition radius, performance drops by more than 20x.

i.e. the receptive fields are kept constant. The approach we propose is to use as many consecutive values in the encoded input sequence as the number of scale-out zones. In this example, we will use 4 encoders to simultaneously encode four different iterations from the input sequence. Thus, we not only increase the throughput of the system but also the load on each individual mini-column, since a whole representation is scattered throughout the whole system. Additionally, increasing the number of zones will keep the total proximal traffic constant (since receptive field size is kept constant).

In an *n-zone* system, each mini-column only sees an *n-th* part of the input data. Therefore, each CC will require an *n-th* part of memory requirements, improving the system scalability. Access times are faster and the time available to accommodate the computation (during the communication phase) is n-times greater, which might allow slower but denser memory technology to be used.

5 Evaluation Methodology

5.1 Tools and Benchmarks

We have developed an integrated simulator, CortexSim[47], which simulates the previously depicted mechanisms. CortexSim is influenced and verified compared to the NuPIC white paper implementation (but with mini-column boost) of the CLA algorithm using the Numenta Anomaly Benchmark (NAB) [35]. The simulator is connected to a network simulator, Topaz [48], in order to obtain precise network timing results and DSENT[49] and CACTI[43] to estimate area and energy requirements. The data sets used in this evaluation are both synthetic and real. We use synthetic data to simplify architectural comparisons and realistic data to provide a notion of the benefits of the accelerator in a practical scenario.

The real data is provided by NAB. The NAB corpus of 58 time series data files, composed of ~350,000 samples, is designed to provide data for research in streaming anomaly detection algorithms. It is comprised of both real-world and artificial time series data containing labeled anomalous periods of behavior. Most data come from real-world scenarios from a variety of sources such as Amazon Web Services metrics, Twitter volume, advertisement clicking metrics, traffic data, and more. The data includes anomalies that are annotated by human reviewers, following a strict procedure. This data is processed using many anomaly detection mechanisms and serves to compare with CLA in this specific task. Each data set has a probationary period (~10%), during which the detector's anomaly detections are ignored. Note that in each time series the detector its reset. Although the data diversity is quite high, the parameters of the cortex are constant in all experiments. Under these conditions, NuPIC can reach 65% successful anomaly detection whereas other state-of-the-art approaches are 20% behind.

For the synthetic workload, we will use periodic series of 32-bit integer data generated from randomly defined polynomials (up to fourth degree with randomly chosen coefficients). A limited number of points from each one is defined for twenty values of *x*, defining a temporal set. We will repeat each temporal set until it is learned by the system. We consider that the time series is learned when the number of elements in the sequence with all the active mini-columns correctly predicted (i.e. no mini-column bursts) is equal to half of all the data points. The rationale of this is to keep half of the time for learning new sequences and half of the time for predicting them. Therefore, half of the iterations will produce the extra traffic of mini-column bursting or low overlap inhibition that a new input sequence appearance will generate. The second half of the time, the system will have a stable representation of the input, which is less demanding for the network. The number of temporal series (i.e. polynomials) used to fulfill strict 98% confidence intervals is around ~50.

In both cases, the classifier we will use is an anomaly score estimator.

5.2 System Configuration

With regard to the CLA configuration, we mimic the one used by [35]: 45x45 mini-column system with 32 cells per mini-column (with up to 128 distal segments), with global inhibition, a 2045-bit SDR encoder with a diameter in the receptive field of 32. In contrast to [35], we use a new SDR encoder, succinctly introduced in section 1.3. This encoder improves NAB anomaly detection by 1-2% compared to the one employed in [35], denoted as Random Distributed Scalar Encoder (RDSE). The encoder, in synthetic workload, has full integer precision. In NAB we limit it to up to 130 levels of quantification (as RDSE does in [35]). All CLA parameters are kept constant throughout the evaluation.

With regard to the network, we employ a 2D Torus topology with a conventional router with deterministic DOR, using bubble flow control [50] as a deadlock avoidance mechanism (single buffer of 160 bytes per port, no virtual channels), 4-cycle pipeline. We assume that the link wires use low-swing links and require a clock cycle to travel from router to router. The clock cycle, conservatively, is set to 1 ns. We use dimension-order replication for multicast deadlock avoidance [39]. The network drain mechanism depicted in Section 4.2 has the routing logic embedded.

6 Performance Results

6.1 Synthetic Benchmark

Figure 8 shows the number of clock cycles required for each input sample, for 11x11, 16x16 and 23x23 tori. (i.e. different numbers of mini-columns per CC). As can be appreciated, for a plain approach (sequential) there is little effect on the network size, i.e. network contention dominates. This is due to the high load that the network supports. Adding more nodes increases the raw bandwidth, which in the case of 23x23, allows the time required to process an input to be reduced slightly. When we add coalescing injectors, the delay is reduced by four times, because the inhibition traffic cannot use the links efficiently. The contention reduction allows this improvement. Adding pipelining opens up the opportunity for further traffic reduction, although limited packet size (80bytes) allows limited aggregation. Nevertheless, the true advantage of pipelining is that computation can be fully overlapped with communication, i.e. we might actually need only around 500 cycles to fully process an input sample.

Since the number of accesses per mini-column is approximately 80 (40 for proximal traffic and 40 for distal traffic), computing might need ~1300 cycles in 11x11, ~640 in 16x16 and ~300 cycles in 23x23. It seems that the most suitable network for this configuration is the 16x16 system.

The reduction in contention of these techniques creates the opportunity to reduce the cost of the network by narrowing the link widths. Figure 9 provides the results of this change.

To improve these figures, the out-scaling zones might be useful. Figure 10 suggests that moving from 1 zone to 4 zones, can significantly reduce the communication cost. It could be possible to process an input sample in ~200 network cycles. This is because the inhibition and distal traffic only has to reach a quarter of the network. Each mini-column on average will perceive a quarter of the remote spikes, therefore the CC will need a quarter of the memory accesses in order to perform the prediction. Consequently, it seems feasible, using a 16x16 system, to achieve ~160cycles.

Figure 11 shows the total power (both active, and leakage) required by the network. In the out-scaled configurations the power is ~250mW for 16x16.

6.2 Realistic Benchmarks

Figure 12 to Figure 15 show how the system will perform with the anomaly detection benchmark. The corpus of the benchmark is composed of different families of data, grouped and tagged on the *x-axis*. The error bars represent the variability of the performance metric within each family. According to Figure 9, a 16x16 with 16B-wide links is the most interesting. This configuration is also the best performer when four out-scaling zones are used. Under this configuration the latency processing each data set is ~500 cycles and just ~300 when out-scaling is used. This means that the 350,000 data of the whole benchmark can be ingested by the accelerator in just 0.175-0.1 seconds. The average power required by the whole system, under these working conditions, will be between 1.2Watts and 350mWatts.

The latest NuPIC implementation in a 2-socket server based on Intel Xeon E5640 running at 2.4Ghz with 60GB of memory requires ~3000 seconds to run the detection phase in a single core and requires approximately 170Watts. Running in 24 hardware threads, it takes 234 seconds and 450Watts. If we take into account the time and power consumed by the accelerator, CLAASIC is between $3 \cdot 10^4$ and $1.8 \cdot 10^4$ times faster. In terms of energy, the efficiency with respect to a single core is between $1.5 \cdot 10^8$ for the single thread execution (versus the more efficient configuration) and $3.4 \cdot 10^6$ for the 24-thread one (versus the less efficient configuration).

Out-scale improvement in speed and efficiency comes at the cost of accuracy. In practice, the average anomaly detection rate falls 9% compared to the standard configuration. Note that none of the remaining parameters of the CLA are changed when out-scaling is used.

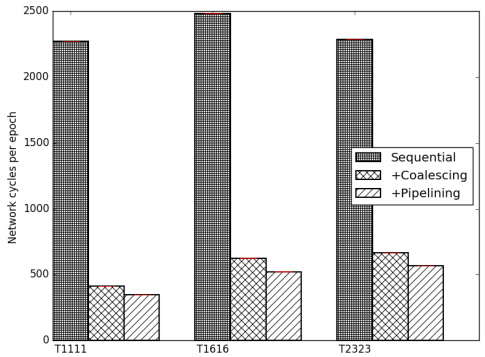


Figure 8 Network clock cycles per input sample for different 2-D square mesh sizes with different algorithm optimizations (16-byte links)

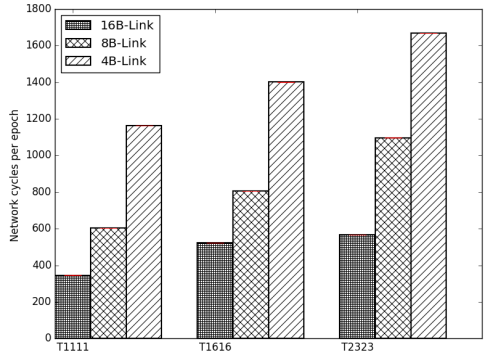


Figure 9 Number of clock cycles per input sample for different 2-D square mesh sizes using pipelining and coalescing injectors with different link widths

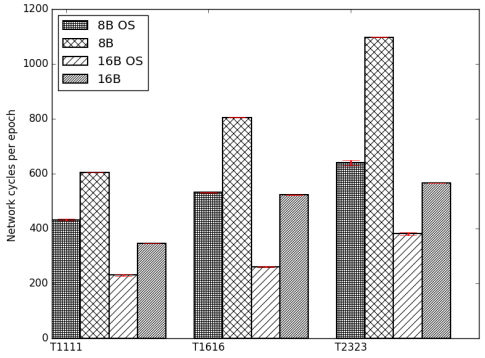


Figure 10 Network clock cycles required to process an input sample with 4 out-scaling zones compared with no out-scaling with 8 to 16-byte links

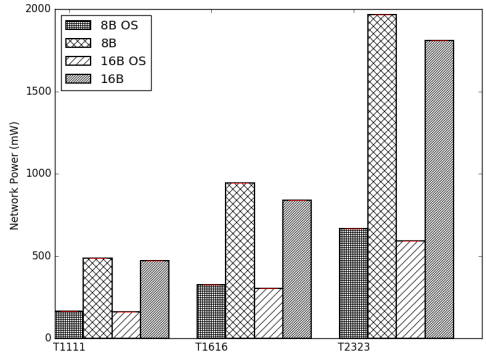


Figure 11 Power required processing an input sample with four out-scaling zones compared with no out-scaling with 8 to 16-byte links

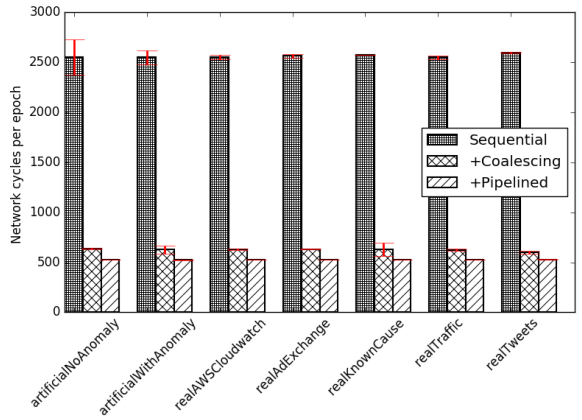


Figure 12 Network clock cycles per input epoch with each NAB data set for a 16x16 CC Configuration with 16-byte wide links, varying optimizations

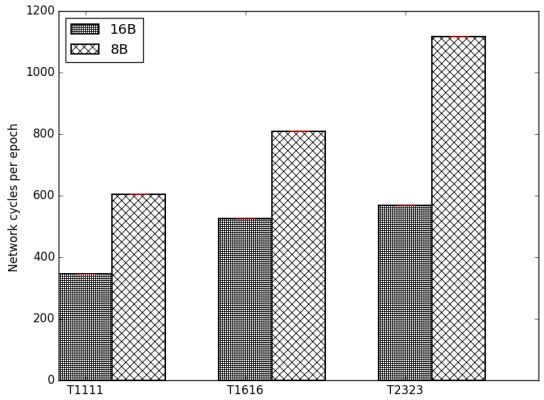


Figure 13 Average Network clock cycles for NAB with different system sizes and network bandwidths

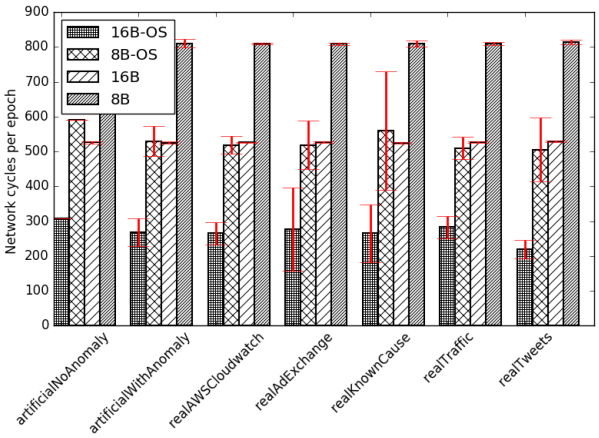


Figure 14 Network clock cycles per input epoch with each NAB data set for a 16x16 CC configuration with and without out-scaling zones

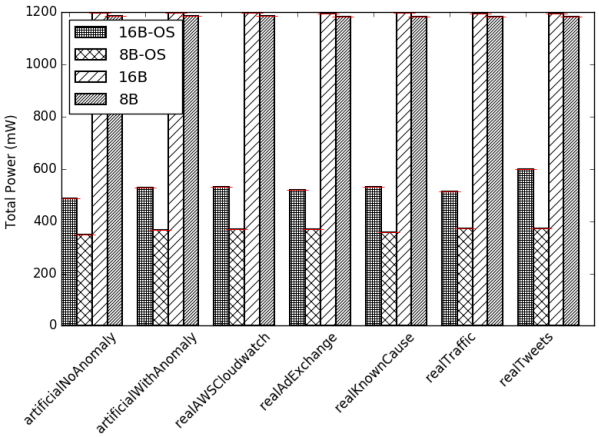


Figure 15 Full chip power requirements with each NAB data set for a 16x16 CC configuration with and without out-scaling zones

6.3 Approximate Power and Area

Although the memory implementation has not been detailed in this work, a rough approximation using a CACTI model for a 256-bank 160MB SRAM (power results have been used in the previous section) was carried out. According to the CACTI and DSENT models, a 16x16 system will require ~43mm^2 (0.154mm^2 per memory and 0.014 per router). Therefore, it seems feasible to scale up the system size without much trouble.

Both energy and power cost can be considered as a worst-case value. On the one hand, the most memory intensive snoops (which correspond to the distal traffic) can be filtered out. On the other hand, CLA fault resilience [16] could be taken into account to minimize the total memory required. In contrast with other approaches, such as [32], we did not need to reduce the number of synapses compared to the software counterpart.

6.4 Constrained Learning Effects on Realistic Benchmarks

Previously, in §3.4, we discussed how in a best-case scenario the learning of a particular mini-column will be 2% of the time. In the case of NAB, the budget time ranges from 10.000 to 30.000 clock cycles. Nevertheless, in this realistic scenario, the activation frequency of the mini-columns is not perfectly homogeneous. To quantify this, in terms of accuracy, we run the simulation imposing a hard date line of x10 in the column-update (i.e. the same mini-column can be updated only in 1 of each 10 input iterations). Therefore, it is not possible to update any dendritic segment with higher frequency (learning is just skipped). The change in the system accuracy as a consequence of this restriction is negligible (<0.1% of changes in anomaly detection in all NAB data sets). Even in this case, still we have a budget of time from 1000 to 3000 clock cycles to perform the learning. With such a relaxed constraint the required hardware can be highly optimized, its cost being negligible in a hypothetical complete design. The only consequence of constraining learning frequency is that the system learns more slowly (which is unnoticed in this application for the time budget set)..

7 Conclusions & Future Direction

This exploratory journey provides a suitable design proposal for a cortex-inspired hardware accelerator. A priori, the solutions presented enable the biggest challenge to be dealt with, namely

the communication substrate. From the evidence gathered, from an engineering standpoint, this is not a problematic issue.

The next steps should address the implementation of learning logic and the dendrite segments. Additionally, the use of other practical problems for the CLA as well as anomaly detection will be considered. Specifically, value prediction might be interesting. For example, combining CLAASIC with a conventional von-Neumann core; we could carry out the classifier task in the regular core, while the CLA algorithm can be executed in the accelerator with a much higher efficiency and speed.

Since CLAASIC multichip organizations are feasible, the proposal is suitable for hypothetical hierarchical organization. Note that inter-region connectivity will be much sparser [51] so it seems practical to fit these communication requirements within the constrained bandwidth of I/O. The fault tolerance resilience of CLA and its low energy requirements, lead us to think that using emerging technologies, such as 3D stacking and non-volatile memories, will be achievable. Consequently, in our view, HTM/CLA might reach biological-level raw capabilities using hardware such as that proposed in this work.

Acknowledgments

This work was supported in part by the Spanish Government (Ministerio de Ciencia, Innovación y Universidades) under Grant TIN2016- 80512-R.

8 References

[1] D. P. Buxhoeveden, "The minicolumn hypothesis in neuroscience," *Brain*, vol. 125, no. 5, pp. 935–951, May 2002.

[2] D. C. VanEssen and H. a Drury, "Structural and functional analyses of human cerebral cortex using a surface-based atlas," *J. Neurosci.*, vol. 17, no. 18, pp. 7079–7102, 1997.

[3] V. B. Mountcastle, "The columnar organization of the neocortex," *Brain*, vol. 120. pp. 701–722, 1997.

[4] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current Opinion in Neurobiology*, vol. 14. pp. 481–487, 2004.

[5] J. T. Wixted *et al.*, "Sparse and distributed coding of episodic memory in neurons of the human hippocampus.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 111, no. 26, pp. 9621–6, Jul. 2014.

[6] I. Theophilou, N. N. Lathiotakis, M. A. L. Marques, and N. Helbig, "Generalized Pauli constraints in reduced density matrix functional theory," Mar. 2015.

[7] G. J. Rinkus, "Sparsey: event recognition via deep hierarchical sparse distributed codes," *Front. Comput. Neurosci.*, vol. 8, Dec. 2014.

[8] S. Ahmad and J. Hawkins, "Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory," Mar. 2015.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[10] F. Clascá, P. Rubio-Garrido, and D. Jabaudon, "Unveiling the diversity of thalamocortical neuron subtypes," *Eur. J. Neurosci.*, vol. 35, no. 10, pp. 1524–1532, 2012.

[11] C. T. Anderson, P. L. Sheets, T. Kiritani, and G. M. G. Shepherd, "Sublayer-specific microcircuits of corticospinal and corticostriatal neurons in motor cortex.," *Nat. Neurosci.*, vol. 13, no. 6, pp. 739–44, Jun. 2010.

[12] F. Briggs, "Organizing principles of cortical layer 6.," *Front. Neural Circuits*, vol. 4, p. 3, Jan. 2010.

[13] A. M. Thomson, "Neocortical layer 6, a review.," *Front. Neuroanat.*, vol. 4, no. March, p. 13, Jan. 2010.

[14] M. Vélez-Fort *et al.*, "The Stimulus Selectivity and Connectivity of Layer Six Principal Cells Reveals Cortical Microcircuits Underlying Visual Processing," *Neuron*, vol. 83, no. 6, pp. 1431–43, Aug. 2014.

[15] M. Lewis, S. Purdy, S. Ahmad, and J. Hawkins, "Locations in the Neocortex: A Theory of Sensorimotor Object Recognition Using Cortical Grid Cells," *Front. Neural Circuits*, vol. 13, no. 3, 2019.

[16] J. Hawkins and S. Ahmad, "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex," *Front. Neural Circuits*, vol. 10, p. arXiv:1511.00083 [q-bio.NC], Mar. 2016.

[17] C. Grienberger, X. Chen, and A. Konnerth, "Dendritic function in vivo," *Trends Neurosci.*, vol. 38, no. 1, pp. 45–54, Nov. 2014.

[18] S. El-Boustani *et al.*, "Locally Coordinated Cell-wide Synaptic Plasticity of Visual Cortex Neurons in vivo," *Science (80-.).*, vol. In Press, no. June, pp. 1349–1354, 2018.

[19] Y. Cui, S. Ahmad, and J. Hawkins, "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding," *Front. Comput. Neurosci.*, vol. 11, Nov. 2017.

[20] J. Mnatzaganian, E. Fokoué, and D. Kudithipudi, "A Mathematical Formalization of Hierarchical Temporal Memory Cortical Learning Algorithm's Spatial Pooler," no. August, pp. 1–11, Jan. 2016.

[21] J. Hawkins, S. Ahmad, and Y. Cui, "A Theory of How Columns in the Neocortex Enable Learning the Structure of the World," *Front. Neural Circuits*, vol. 11, pp. 0–15, Oct. 2017.

[22] M. Pietroń, M. Wielgosz, and K. Wiatr, "Formal Analysis of HTM Spatial Pooler Performance Under Predefined Operation Conditions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9920 LNAI, 2016, pp. 396–405.

[23] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning," Nov. 2015.

[24] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," May 2016.

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] "AlphaGo Wins Final Game In Match Against Champion Go Playe," *IEEE Spectrum*, 2016. [Online]. Available: http://spectrum.ieee.org/tech-talk/computing/networks/alphago-wins-match-against-top-go-player.

[27] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," Oct. 2014.

[28] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[29] A. S. Cassidy *et al.*, "Real-Time Scalable Cortical Computing at 46 Giga-Synaptic OPS/Watt with ~100× Speedup in Time-to-Solution and ~100,000× Reduction in Energy-to-Solution," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 27–38.

[30] T. Chen *et al.*, "DianNao," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, 2014, pp. 269–284.

[31] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," Feb. 2015.

[32] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," Feb. 2016.

[33] W. Sin, K. Haas, E. Ruthazer, and H. Cline, "Dendrite growth increased by visual activity requires NMDA receptor and Rho GTPases," *Nature*, vol. 2112, no. 1998, pp. 2108–2112, 2002.

[34] DeSieno, "Adding a conscience to competitive learning," in *IEEE International Conference on Neural Networks*, 1988, pp. 117–124 vol.1.

[35] A. Lavin and S. Ahmad, "Evaluating Real-Time Anomaly Detection Algorithms -- The Numenta Anomaly Benchmark," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015, pp. 38–44.

[36] N. Caporale and Y. Dan, "Spike Timing–Dependent Plasticity: A Hebbian Learning Rule," *Annu. Rev. Neurosci.*, vol. 31, no. 1, pp. 25–46, 2008.

[37] F. Walter, M. Sandner, F. Rcohrbein, and A. Knoll, "Towards a neuromorphic implementation of hierarchical temporal memory on SpiNNaker," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[38] A. Kumar, Z. Wan, W. W. Wilcke, and S. S. Iyer, "Toward Human-Scale Brain Computing Using 3D Wafer Scale Integration," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 1–21, Apr. 2017.

[39] N. E. Jerger, L. S. Peh, and M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *35th International Symposium on Computer Architecture - ISCA'08*, 2008, pp. 229–240.

[40] J. Duato, S. Yalamanchili, and L. Ni, "Interconnection Networks: An Engineering Approach," Oct. 1997.

[41] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous Online Sequence Learning with an Unsupervised Neural Network Model," *Neural Comput.*, vol. 28, no. 11, pp. 2474–2504, Nov. 2016.

[42] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: a cheap and robust fault-tolerant packet routing mechanism," in *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, 2004, pp. 198–209.

[43] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 3–14.

[44] R. M. Bruno and D. J. Simons, "Feedforward mechanisms of excitatory and inhibitory cortical receptive fields.," *J. Neurosci.*, vol. 22, no. 24, pp. 10966–10975, 2002.

[45] R. Lorente De Nó, "Studies on the structure of the cerebral cortex. II. Continuation of the study of the ammonic system.," *J. für Psychol. und Neurol.*, vol. 46, pp. 113–117, 1934.

[46] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, Jun. 1996.

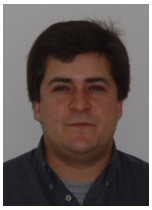
[47] V. Puente, "Cortexim." [Online]. Available: https://github.com/cortexsim.

[48] P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente, and J.-Á. Gregorio, "TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 99–106.

[49] C. Sun *et al.*, "DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," *Proc. 2012 6th IEEE/ACM Int. Symp. Networks-on-Chip, NoCS 2012*, pp. 201–210, 2012.

[50] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. M. Prellezo, "The Adaptive Bubble Router," *J. Parallel Distrib. Comput.*, vol. 61, no. 9, pp. 1180–1208, 2001.

[51] S. M. Sherman, "The function of metabotropic glutamate receptors in thalamus and cortex," *Neuroscientist*, vol. 20, no. 2, pp. 136–149, 2014.



Valentin Puente received the BS, MS and PhD degree from the University of Cantabria, Spain, in 1995 and 2000 respectively. He is currently an Associate Professor of Computer Architecture at the Department of Computers and Electronics of the same University. His research interests focus on computer architecture and the impact that emerging paradigms or technology changes might have on it.

Jose-Angel Gregorio received the BS, MS and PhD in Physics (Electronics) from the University of Cantabria, Spain, in 1978 and 1983 respectively. He is currently a Professor of Computer Architecture in the Department of Computers and Electronics in the same University. His main research interests focus on chip multiprocessors (CMPs) with special emphasis on the memory subsystem, interconnection network and coherence protocol of these systems.