Seminar High Performance Computers - Current trends and developments
Winter Term 2015/2016

Got Brain(s)? A Review and Critical Assessment on Neuromorphic Computing

Daniyal Kazempour
Ludwig-Maximillians-Universität München

03.12.2015

Abstract

The current progresses made in developing new computing systems excel in their computational power and energy efficiency. In this review we introduce the new emerging architecture of neuromorphic computing systems. We take a closer look at the structure of the neuromorphic chip, with specific focus on the neurosynaptic core as well as the foundations of the event-driven paradigm. We introduce the components of the Corelet ecosystem, where a Corelet is a programming model designed for neuromorphic computing. Further we take a look on a few selected algorithms and applications. Based on the results of the benchmark, a critical assessment follows. Finally this review is concluded by some final remarks and future prospects.

1 Introduction

In the beginning we will motivate the topic by giving a brief introduction of the todays predominant computing architecture. Further we motivate by stating the core aims that drive the development of the neuromorphic computing architecture: power consumption and the degree of parallelism. As a next step the elements from which this architecture is inspired will be explained. In order to set the foundations for the following sections an brief introduction in the event-driven computing paradigm is provided as well as a comparison of two neuromorphic computing architectures.

1.1 The von Neumann architecture and its limitations

In this section we are going to introduce the von Neumann architecture and further describe its limitations. The von Neumann architecture is the one design which gives shape to the computing architectures since the mid of the 20th century. The architecture consists of a central processing unit (CPU) which itself consists of a control unit (CU) and an arithmetic logic unit (ALU). The ALU performs the computation and logic linking while the CU interprets the instructions of a program and wires up several elements such as data source, required ALU components etc. Further the architecture bares a memory and an I/O unit.

As a major drawback of the von Neumann architecture, data operation and instruction fetch can not be performed at the same time due to the fact that they both share a commonly used bus. This cir-

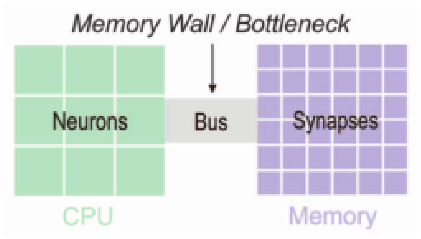

Figure 1: The von Neumann architecture bottleneck in context of neural structures [11]

cumstance is also named as the *von Neumann bottleneck* [3]. Where the von Neumann architecture specifies only one memory component the neuromorphic chips architecture intends that every node has its own local memory and a dedicated private memory bus.

1.2 Degree of parallelism and power consumption

In this section we are going to describe the increasing importance of parallel computing. In 1965 Gordon Moore stated an observation that the number of transistors in an integrated circuit doubles approximately every two years, known as Moores Law [12]. This law however can no longer be maintained by adding more transistors to a chip, due to the fact that the integrated circuits are eventually reaching a certain density where natural physical limitations arise. As a consequence a higher degree of parallelism is required.

One technology that aims to saturate the need for parallelism is GPUs. GPUs rely on the single instruction multiple data (SIMD) data parallelism model. The architecture of GPUs can be assigned to the category of vector processors. These types of processors aim to perform an operation on large data sets in parallel. Instructions are given to load a certain amount of data, then perform the operation. While scalar processors operate on single elements (scalars), vector processors operate on linear sequences of numbers. Exploiting those properties GPUs aim to enable a high degree of parallelism just like neuromorphic chips.

The previously explained von Neumann bottleneck limits the effectively exploitable degree of parallelism. This obstacle is aimed to be reduced in the neuromorphic computing architecture as we shall see further in this paper. In this context it may be worth to emphasize the importance of parallelism in today's computing. Until about 2005 (ref: top500.org) the aims in computing power lay within the idea of increasing the clock speed of a system. However within the last decade the potential of parallelism has been further discovered, especially accelerated through the fields of simulation and data-driven analysis of large data volumes which is more widespread known under the buzzword *big-data*. In the big-data domain, one method heavily relies on parallelism: deep learning.

Deep learning is, simply put, a neural network employing a large cardinality of layers. A neural network itself is a bio-inspired machine learning method for classification and prediction problems. As we shall see in the next section, neural networks resemble their biological counterparts, massively relying on parallelism. In order to saturate the need of parallelism, currently neural networks make heavy use of accelerator hardware such as graphics cards and their GPUs or accelerator cards such as the Intel Xeon Phi. The concept of neuromorphic computing also aims to serve a suitable architecture to achieve a high degree of parallelism for deep learning systems.

While computer systems, especially the supercomputers increase in the degree of parallelism, another trend that became visible in the past years is the reduction of power consumption. As computers get more powerful, the power consumption also increases, with financial and environmental consequences. According to spectrum.IEEE.org the True North neuromorphic chip has a power density of $20mW/cm^2$ which would be equivalent to $1/10,000$ of the power density of modern microprocessors. However these values have to be taken with a grain of salt as the performance of the von Neumann and

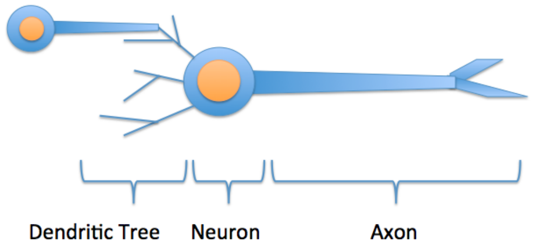

Figure 2: Neurons and their signal propagation

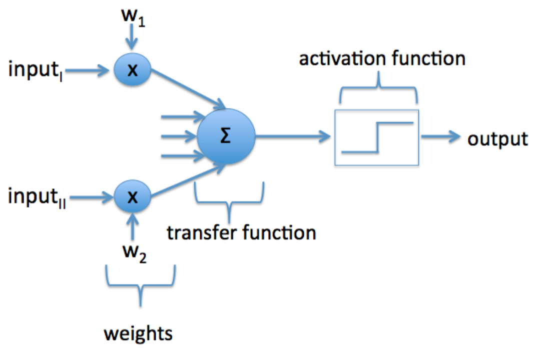

Figure 3: Functional model of neurons

the neuromorphic architecture cannot be directly compared. In a later section, a critical assessment of the benchmark results is discussed and we will discuss how performance measures have to be evaluated.

1.3 Biological neuron-model

In this section we are going to take a closer look at the details of the biological model giving rise to the neuromorphic chips. The previously mentioned concept of neural networks as well as the neuromorphic computing architecture were not being designed completely without any element which inspired them. The source that gave the idea to both concepts can be found in the domain of biology. Various organisms which possess a nervous system have so called neurons which pose the fundamental unit of every nervous system. Its structure as it can be seen in Figure 2 as well as its functional model [1] is as follows:
The specific biological model which is relevant to both, neuromorphic chips and artificial neural networks is the neuron spiking model. It consists of dendrites and axons. The dendrites which are heavily branched receive the excitation from other neurons. Therefore the dendrites can be regarded as input-lines of a neuron. The output of a neuron is sent through the axons which lead to another neuron. The signals that are sent through the axons are encoded through their voltage difference. This difference is achieved through a mechanism in which ion-streams are permitted to pass, where this mechanism is realised through specific ion-channels on the cell-membrane. At the end of the axons which lead to another neuron, the signal is conducted via chemical gradients which are achieved via vesicle based neurotransmitter release.
Neurons fire in the sense of releasing a signal. This phenomenon can be observed if a spatial and temporal summation of different changes in the membrane potentials from various cells which direct into a neuron exceed a certain threshold potential. If this is the case, then an action potential is created and sent via the axon to another cell which can be seen schematically in figure 3.

1.4 von Neumann vs. neuromorphic chips

In this section we are going to compare the von Neumann architecture and neuromorphic chips. However we are not going to compare to the neuromorphic chips directly. We rather compare here to the architecture on which the neuromorphic chip actually relies on: the so called connection machine [7].

The connection machine of Hillis aimed to develop a computer with the structure of the brain. High performance should be achieved via many simple processing elements (PEs) and via a high connectivity. This resulted in certain design decisions such

[1]The perceptron. A probabilistic model for information storage and organization in the brain. In: Psychological Reviews, 65 (1958): S. 386-408.

as the usage of e.g. a hypercube as the topology of choice. The hypercube contained four sub cubes, which on their own included around thousands of simple PEs that were connected with each other. Each of the PEs is connected to its four next neighbours. By this a grid structure emerges in which each PE can be exactly located via a 12-bit address. Compared to the von Neumann architecture the focus here lies in the high degree of connectivity and less on the processors. Another unique feature of the connection machine is the fact that each PE has its own local memory of 4 kBit, a flag-register with 8 Bit and an ALU. The programming of the system is performed via a specific LISP dialect.

Another major difference between the Von Neumann and the neuromorphic computing architecture is that the latter is not driven by a certain CPU clock rate but is rather event driven. This means that each core is acting based on an incoming event in a sense of e.g. an arriving signal. Noteworthy in this context is that the biological and engineering counterpart are also event-driven systems.

1.5 Two architectures in comparison

There are currently several neuromorphic computing projects that developed their own neuromorphic architectures. Describing all of the developed systems would be beyond the scope of this paper, therefore we are going to describe two of them here. One of them is the True North system developed by IBM,DARPA,SyNAPSE and another one is Neurogrid developed by the Stanford university.
The major differences between both architectures are that Neurogrid is an analog/digital hybrid model, whereas True North is completely digital. The terms of analog and digital are meant in a way that the neurons are implemented as analog circuits via wires and meter charges, and digital neurons are implemented by using digital elements such as RAM, counter and comparators as it can be seen in figure 4. Further True North consists of an hierarchical design and contains neurosynaptic cores as their basic building blocks, which we shall describe in detail in the upcoming section.

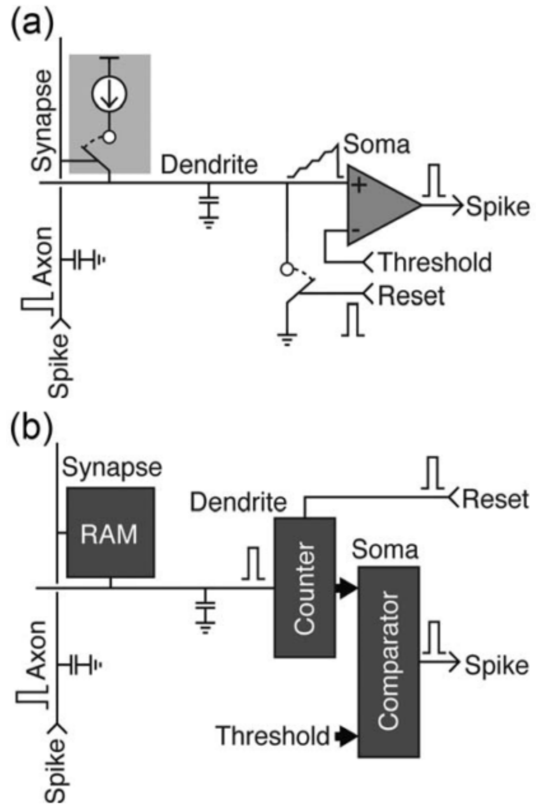

Figure 4: (a) analog and (b) digital implementation of a neuron [4]

Besides these mentioned major differences there are also some differences in the number of neurons, where True North provides 16 million neurons and 4 billion synapses in its current configuration, the Neurogrid provides only one million neurons and with 8 billion, twice as many synapses as True North. The certainly most interesting distinctive property are the project aims of both systems. Where True North aims for low-power neuromorphic chips which are designed for applications in mobile sensors and cloud computing etc., Neurogrid is created with the purpose to simulate biologically realistic brain models, although the team has further moved on to the development for autonomous robot applications.

2 The neurosynaptic core

This section begins the bottom-up approach of showing the structural and functional units of the neuromorphic computing architecture, starting from the neurosynaptic core which represents the atomic unit of the neurosynaptic chip as it can be seen in figure 5.

2.1 Architecture Model

Recalling the section in the introduction which described the biological counterpart, the neursynaptic core is composed of axons, synapses and neurons as it can be seen in figure 6. Here it can be seen that information is directed from the axons to the corresponding neurons. This control of the information flow from axons to neurons is ensured via synapses. Taking a closer look at figure 6 we see a representation of the mentioned units in a grid-like structure. This structure is composed of K axons that are linked through KxN binary-valued synapses to N neurons. In figure 6 we can see that the rows represent the axons, while the dendrites represent the columns. The row-column junctions represent in this grid-view the synapses which is coined with the term *neurosynaptic crossbar*.

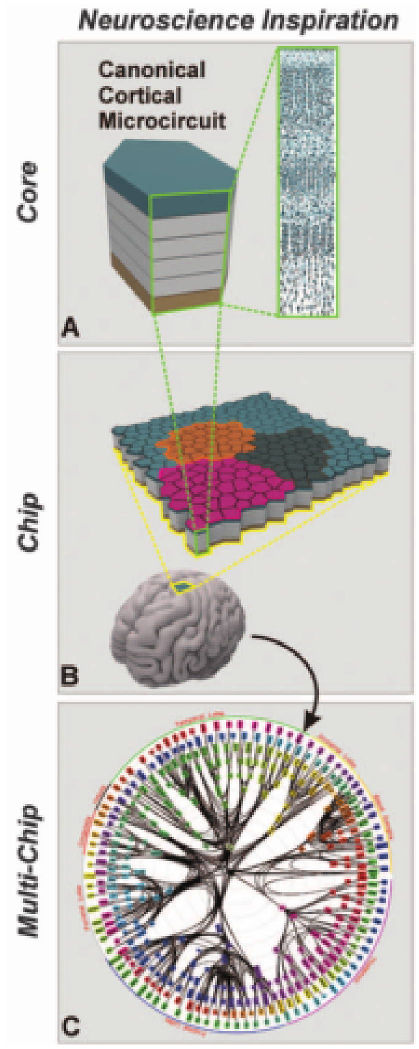

Figure 5: From the neurosynaptic core to the neurosynapic computer [11]

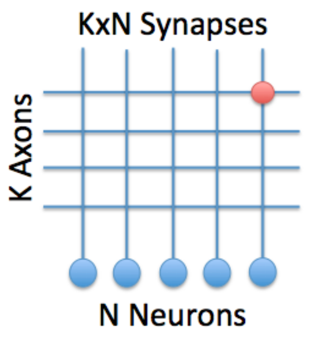

Figure 6: The architecture of the neurosynaptic core [10]

2.2 Operation

The model according to which the neurons operate is the *leaky integrate-and-fire model.*
The membrane potential of a neuron is updated by taking its current membrane potential adding to it the leak of the neuron and finally adding the sum of the inputs from all axons the neuron receives.
If the newly computed membrane potential of a neuron exceeds a certain threshold θ then the neurons produce a so called spike which can be compared to the analogy that a neuron fires. At the same time where a spike is created, the potential of this neuron is set back to 0. The resulting spike can be compared in a way to an binary flipping.

2.3 Event-driven Implementation

In context of event-driven computation the previously mentioned connection machine follows a different programming paradigm model in contrast to the linear sequential programming model of von Neumann [1]. Driven by that statement the question which may emerge is: How is a program actually stored in neuromorphic chips/the connection machine compared to the von Neumann architecture. In von Neumann programs are stored via their assembly opcodes. In contrast in the neuromorphic chip/connection machine programs are stored through their connections. In the connection machine for example, the concurrent operations in Cm-Lisp, a Lisp dialect for the connection machine are implemented via a data structure called a xector, which is an abstraction of a set of processors with a value stored in each processor [7]. In the neuromorphic architecture upon execution of a corelet a videoToSpikes function is triggered in order to generate an input spike file. The generated spikes are further mapped to a core-axon tuple. This process named transducing, compares to what is known in the von Neumann architecture as compiling [1].

Having taken a look at the neurosynaptic core architecture and the way it operates, we will now further describe the event-driven implementation of this architecture. Here the authors faced challenges by finding meaningful tradeoffs with regards to power consumption, performance and density, where the focus is set on the first two attributes. Achieving a reduction in power consumption the neural updates are performed in an event-driven way. By giving a dedicated circuit per neuron a high degree of parallelism is achieved, although this level of parallelism comes with the prices of inefficient density. With density we refer here to circuits on the chips that are unused as stated in [8].

Taking these goals into account the authors have developed a block-level implementation of the neurosynaptic core which is composed of an input decoder with 1024 axon circuits, a 1024x256 SRAM crossbar, 256 neurons and an output encoder as it can be seen in figure 7. The way the communication is performed at the input and output interfaces to the core is achieved through an address-event representation (AER). The AER encodes binary activities like e.g. $A(t)$ by sending the location of the active elements through a multiplexed channel.
Unfortunately no further informations are provided on how TrueNorth performs collision handling, given access to a shared medium. The only information that is given in [11] mentions that spike events are carried out between cores via time-multiplexed wires.

Each operation per time step that is performed on a core is composed of two phases, where the first

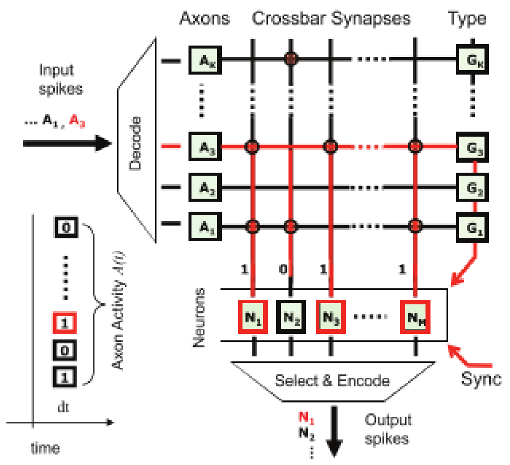

Figure 7: A look on the internal blocks of the neurosynaptic core [10]

phase realized the axon-driven component while the second phase takes care of the time step synchronisation. Taking a closer look at figure 7 it can be seen that in the first phase spikes in form of address-events are sent to the cores one at a time and are further sequentially decoded to the according axon block. If an axon has received an event the entire SRAM row is activated by the axon. By this activation all connections of the axon and their types are read out. Where ever a connection is existent (encoded by a 1) the signals are sent to their neurons on which the membrane potentials are updated. Having updated all neurons, the axon stops its reading process and awaits the next address-events. The procedure is repeated until no more address-events are left.
In the second phase a synchronization of all neurons is ensured by sending a synchronization event once every millisecond. The synchronization of the neurons implies that each neuron checks if its membrane potential has exceeded the threshold θ. Given the case that the threshold is reached, a spike is produced and sent as address-events as described in phase one and finally a reset of the membrane potential to zero is performed. Here it may be already recognized that the two-phase mechanism is required in order to ensure a lock step of hardware and software at the end of each time step.

3 The machine model

In this section the von Neumann and the neurosynaptic machine model are compared with a special emphasis on the differences.

The von Neumann machine model is characterized, to put it in a nutshell, through its instruction cycle of fetching instructions, decoding instructions, evaluating the addresses, fetching operands, executing the instructions and storing the results by write-back in memory data. The key elements here are the opcodes and the instruction pointer. The von Neumann machine model itself is highly dynamic in the sense that due to the mutability of the opcodes on the memory, states can be defined, leading to a system where several states can change over time.

In contrast to von Neumann, the neurosynaptic machine model does not consist of opcodes but of spikes. The input and output spike configurations are stored in the local dedicated memory of the neurosynaptic core. To be more precise a connection matrix and a weight vector are the stored elements. Due to the fact that connections among the cores are highly static and that the model is event-driven, leads to the consequence that the neurosynaptic machine model is not directly mutable like the von Neumann architecture, implying that it is not a state-based system but rather a pure functional system where every input leads to an output, without any states being stored in between. This implication bares also the fact that without states, the system is free of side-effects. The static character of the neurosynaptic machine model however implies that no neuroplasticity is provided. In a system where neuroplasticity is ensured, the connections are not static, but would be capable of changing over time.

The lack of states in the neurosynaptic machine model however does not imply that the model would not be turing complete [2]. However as Alonzo Church has shown the turing completeness of Lambda-calculus in the sense of functional programming, the authors state that the neuromorphic model is also turing complete. This is achieved through the fact that with the neurosynaptic cores logic gatters can be built which in turn can be used to create a turing machine.

4 The Corelet ecosystem

In this section we elaborate how neuromorphic chips can be programmed by using Corelets. So far we have described the motivation and inspirations that lead to the neuromorphic computing as well as its neurosynaptic core with its architecture and implementation. In order to use, in the sense of programming the system a paradigm is required that meets the needs of the True North architecture.

4.1 Corelets

So far we have dealt with neurosynaptic cores, the most atomar units of the neuromorphic computing architecture. If we now want to deal with a *network of neurosynaptic cores* the so called *Corelets* can be used. Corelets are a programming model based on signal flow between logical subsystems. Operations on a subsystem are composition, connection and decomposition. Corelets provide an abstraction of a network of neurosynaptic cores with the aim of encapsulating the intra-network connectivity while exposing only external inputs to and outputs from another network. The grouping of inputs and outputs is performed within so called *connectors*. In figure 8 it can be seen how a neurosynaptic core is further enhanced by connectors . The construction is abstracted by a seed Corelet. We have here a black-box like model where the user only needs to know what is put inside the Corelet and to see what comes out of the Corelet. A Corelet A can now be linked with a Corelet B to a Corelet C in a Lego-fashioned manner as it can be seen in figure 8. This process is known as a Corelet *composition*.

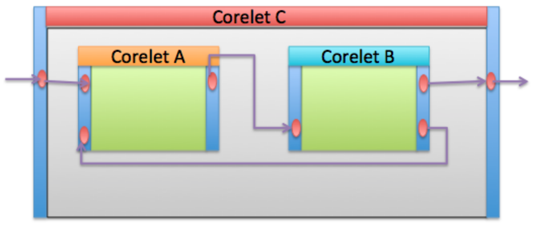

Figure 8: Seed Corelets and a composition of Corelets [1]

The composition can be performed hierarchically to design more complex Corelets. The authors of the paper refer to Corelets being regarded as trees of subcorelets. However another possible more intuitive presentation of Corelets would be their recursive character in general, regardless of any tree or list like structure. In order to directly implement a written program on the True North hardware it is required to *decompose*. The process of a Corelet decomposition poses the inverse of a composition. By applying decomposition, all encapsulation layers are removed which reveals a network of neurosynaptic cores that can be now implemented on the True North system.

4.2 Corelet language

Building on the Corelet abstractions a language is designed which consists of fundamental symbols such as the neuron, neurosynaptic core and a corelet. However, regarding the tradition of language theory, to any language there is also the need of a grammar which is provided by the connectors of the Corelets. Both, the language and the grammar set the foundation for a sufficient expressiveness of a True North program.

In listinig 1 one example of a Corelet program for music composer recognition (MCR) can be seen. The example shows the object-oriented nature of the Corelet language as it can be seen in Line 1 where the program inherits from the corelet class, and line 8 and 9 which show a method and a constructor. Also the connectivity.

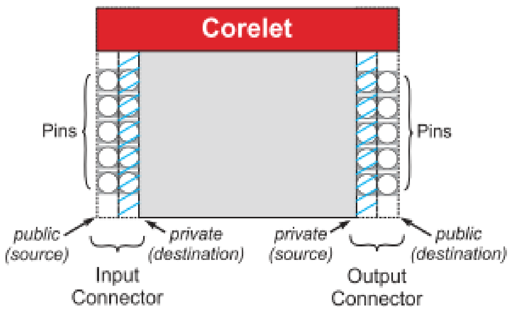

Figure 9: Connectors in an Corelet [1]

Listing 1: A code listing use case for a music composer recognition program [1]

```
classdef MCR < corelet
  %Corelet MCR(nInputs, nLayers,
     nClasses, W)
  %nInputs - number of inputs per
     layer
  %nLayers - number of layers in
     LSM
  %nClasses - number of Music
     Composers
  %W - classifier weight matrix
  methods % public
      function obj = MCR(nInputs,
         nLayers, nClasses, W)
      obj.name = 'Music Computer
         Recognition Corelet';
      %create sub-corelets
      lsm = LSM(nInputs,nLayers);
      sc = SC(nClasses,W);
      obj.subcorelets = [lsm,sc];
      %create connectors
      obj.inp(1) = connector(
         nInputs, 'input');
      obj.out(1) = connector(
         nClasses, 'output');
      %connect sub-corelets and
         connectors:
      obj.inp(1).busTo(sc.inp(1));
      lsm.out(1).busTo(sc.inp(1));
      sc.out(1).busTo(obj.out(1));
    end %of constructor
  end % of methods
end % of classdef
```

The Corelet language has been realized using object-oriented programming (OOP) concepts. The authors motivate this decision with three requirements of Corelets. In the Corelets we have an encapsulation of what is between the input and output, this is one of the core features of OOP. For this figure 9 shows the encapsulation among the connectors in private and public which may be familiar from the modifiers in some OO programming languages. Another requirement is that all Corelets have to use similar data structures and operations and need similar ways of access. This leads to the feature of inheritance which is provided in the OOP. And the last requirement lies within the decomposition of Corelets where each operation is named homogeneously across multiple corelets yet can be heterogeneous among the Corelets regarding their definition. This circumstance requires a feature such as polymorphism. However, all these mentioned features can also be implemented in mostly functional based programming languages such as e.g. Lisp where object-orientated paradigms are provided, yet the principle of higher-order functions is maintained.

4.3 Corelet Library

Like many programming languages the Corelet language comes with a library. This library is a repository of functional primitives which can be taken as ready-to-use building blocks. The current library includes Corelets for simple applications such as scalar functions, algebraic, logical and temporal functions as well as advanced tasks such as linear filters, kernel convolution, finite state machines up to Discrete Fourier Transform, Restricted Boltzmann Machine etc. In this case it would be of special interest to see if any neural network Corelet is implemented which would serve the purpose for deep learning applications.

5 Compiler

In this section we are going to show the way compiling is performed on the neuromorphic architecture. For this purpose we shall first describe the compilation process briefly in order to motivate and further emphasize the differences between the von Neumann and the neuromorphic architecture.

On a von Neumann machine source code is sent in a source file to the preprocessor. This generates a preprocessed source file, which is further sent to the compiler. The compiler generates an assembly file which is forwarded to the assembler creating an object file. The object file, is further sent to the linker which generates an executable.

On the neuromorphic computing system the process of compiling is different as we elaborate in the following lines:

As it can be seen in figure 10 a given program in the form of a Corelet generates upon execution a video file. This video file with an input map file is taken to the location at which each frame of the video spikes are extracted through the individual pixel gray levels. The spikes are then further mapped to a core-axon tuple by using the input map file. Converting data, in this case from a video to spikes is coined with the term *transduction*. The transduction process generates an input spike file. This file is taken with a model file and a configuration file to an external simulator. The simulator generates as an outcome of this compiling process an output map and an output spikes file. The information in these files is then stored in the local dedicated memory of the corresponding cores.

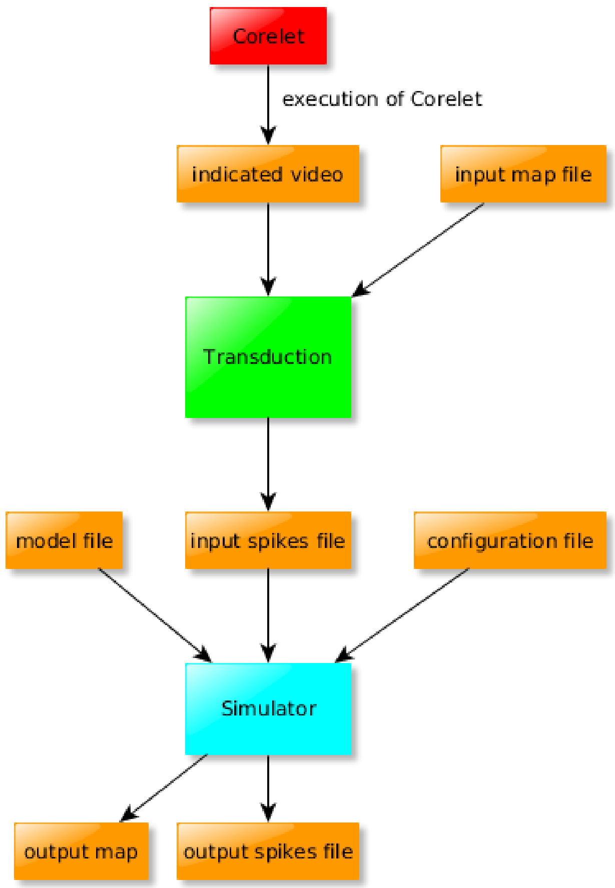

Figure 10: The compiling process on the TrueNorth

6 Algorithms and Applications

In the previous section we have mentioned that the Corelet library provides several ready-to-use modules such as linear classifiers, linear regression etc.

In this section we take a look at one of the application that is provided by the library - the hidden markov models.

6.1 States in a stateless model

In most of the cases when a problem can be reduced to the detection of patterns within a sequence (of symbols, spoken words, events etc.), hidden markov models (HMM) provide a robust choice in order to achieve a solution. The reason of having chosen HMMs is due to one significant property: They are heavily relying on states. How is it possible to express states in a machine model which is completely stateless as described in previous sections?
As a brief recap we describe that a HMM is a probabilistic model in which a system is described through Markov chains with unobserved states. In a more formal definition an HMM is a 5-tuple $\lambda = (S; \Sigma; T; E; \pi)$ consisting of:

- A set of all states S the model can take
- The alphabet Σ of all possible observations
- A transition matrix T describing the transition probability changing between the states
- An emission matrix E giving the probability to observe in a given state s_i a specific alphabet $v \in \Sigma$
- An initial distribution π indicating the probability that a given state s_i is the initial state

Considering this definition an HMM can be regarded as an special case of a dynamic Bayesian network and as an finite state machine. Here we see the massive state-relying nature of HMMs.
The authors in [6] make use of HMM on a TrueNorth system for the following problem: An English sample text is given, the goal is to classify the sequence of characters into the two classes consonants (s_1) or vowels (s_2). For training the HMM the forward/backward algorithms have been applied on a sequence of 30000 characters from unlabeled English text.

In order to implement the desired HMM, as described in the previous section, Corelets were composed together. The HMM is composed of four Corelets which shall be further described:

- Observation Probability: The observation probabilities are acquired in this Corelet by using neurons with stochastic synapses. The probability of the synapses in this Corelet to deliver incoming spikes is done stochastically. The threshold θ of the neurons is 1 and one synapse encodes one state.
- State Computation: The computation of the probability that the system is in a state s_k at time i is performed by this Corelet which uses three neurons per state for the mentioned computation. There are two input neurons that compute the probability that the HMM will end up in each state, and a third one which sums up the output of the two input neurons, leading to a spike at the output.
- State Memory: In order to store the previous and current states of the HMM so called rate store neurons are used within this Corelet. The very characteristics of these neurons lies in their configuration via a stochastic threshold and the absence of leak. This leads to the property that the output spike train of such neurons encodes the neurons membrane potential value in a probabilistic manner.
- WTA (winner-take-all): This Corelet returns the maximum-likelihood estimate of the class where each letter belongs to at each time segment.

The authors in [6] aimed to evaluate the HMM which lead to an accuracy of 99.8%. In total 38 neurons were used. It is further stated that in the implementation the number of neurons that were needed scale linearly with the number of states modeled in the HMM. In conclusion the authors state that HMMs benefit from neuromorphic chips regarding the higher accuracy and linear scaling of the number of states with the number of neurons. However some counterpart for comparison is required in order to give the accuracy its expressiveness. Besides the point of lacking comparability, the benchmark of further HMM applications would be required in order to be able to make a statistically profound statement on the true benefits of neuromorphic chips for HMMs.

7 Critical assessment and concluding remarks

In this section we provide concluding results and aim to give an answer to questions such as if neuromorphic chips are a viable alternative, if they are restricted to specific problem domains and if there are already working prototypes. Taking all aspects discussed in this paper into respect, it appears that many of the features that neuromorphic computing provides are not such a novel approach as it is motivated in the various papers. Neuromorphic chips are an application of the research on neuromorphic *engineering* which has been first realized by Carver Mead in the late 1980's [9].

Also the idea of encapsulating the internals of a neurosynaptic core in form of Corelets is not an novel approach. It is merely applying the black-box approach to the neurosynaptic cores which is already visible at e.g. neural networks. Another aspect which has been mentioned however not been sufficiently explained in detail is the Turing-completeness of the TrueNorth architecture. While in [10] it is stated that the architecture satisfies the property, it leaves this statement, concluding that it would be straight-forward to show that the architecture is actually Turing-complete.

In this context it is vital to emphasize that the neuromorphic computing architecture does indeed *emulate* the functionality of neurons while e.g. an artificial neural network (ANN) *simulates* the behavior of neurons. While in a simulation the underlying state of a system is in the focus to be modeled, in an emulation a system is mimicked without any detailed and accurate reflection of the internal states. An emulation aims to substitute the system it is emulating, while a simulation serves the purpose for further study and analysis. The TrueNorth architecture aims to emulate the functionality of the brain.

Further, if we take a look on how the performance of today's systems are measured, we get confronted with the FLOPS unit (floating point operations per second). In the domain of neuromorphic computing the so called SOPS unit (synaptic operations per second) is used. Given this fact, the question may emerge in how far those two units of performance measure are comparable. Given the case that no comparison between FLOPS and SOPS is possible, how can the performance of a neuromorphic computing architecture be compared to the von Neumann based computers? The papers related to the TrueNorth architecture lack further details on if and how SOPS can be compared to FLOPS.

Despite all the criticized aspects, the facts remain that neuromorphic computing provides a different architecture compared to those of von Neumann. Also the energy efficiency is a certain advantage over today's conventional systems. This advantage is of special interest, which gets visible by taking a look on the green top 500 computers (http://www.green500.org) which lists the most energy efficient supercomputers currently in use. The first 10 ranks in the list are all systems which heavily rely on accelerator cards such as e.g. Intel Xeon-Phi or Nvidia cards. These accelerator cards provide a high level of parallelism while being low on power consumption. Neuromorphic chips may have the potential to take this trend of high parallelism while having low power consumption to a new level.

However further research, and development on the architecture as well as understanding of the human brain is required in order to significantly improve this architecture. Aspects such as neuroplasticity are not covered yet by the True North system. This statement can be derived from the fact that the neuromorphic chip architecture does not provide options to have feedback loops on hardware-level leading to mutual-influence of the neurosynaptic cores. This property goes hand in hand with the stateless architecture. In biological sense we understand under neuroplasticity the ability of a neural system to change over time in a sense that e.g. existing synapses degrade while new synapses are established depending on the learning processes that are performed. So far no ability of dynamically transforming existing synaptic structures is existent. What could be also of potential interest, is the observation that none of the currently developed neuromorphic systems consider to mimic the saltatory excitation conduction as observed in the nervous system of vertebrates. It would be comparable to fast-lanes on the highway, enabling certain computation tasks to be sent with higher speed and priority to the destination neurons. However, this remains pure theory and would require a substantial amount of research in order to be determinable if it provides any true advantages to the neuromorphic architecture.

Therefore this concept remains as for now as an experimental approach such as the adiabatic quantum computing. Having matured over time neuromorphic computing may get feasible to be applied as e.g. accelerators in addition to conventional computers, specially in the field of deep learning.

References

[1] Arnon Amir, Pallab Datta, William P Risk, Andrew S Cassidy, Jeffrey A Kusnitz, Steve K Esser, Alexander Andreopoulos, Theodore M Wong, Myron Flickner, Rodrigo Alvarez-Icaza, Emmett Mcquinn, Ben Shaw, Norm Pass, and Dharmendra S Modha. Cognitive Computing Programming Paradigm: A Corelet Language for Composing Networks of Neurosynaptic Cores. *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN 2013)*, 2013.

[2] A.M.Turing. On Computable Numbers, with an Application to the Entscheidungsproblem - A Correction. *Proceedings of the London Mathematical Society. 2*, 42(2):230–65, 1936.

[3] John Backus. Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[4] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R. Chandrasekaran, Jean Marie Bussat, Rodrigo Alvarez-Icaza, John V. Arthur, Paul a. Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[5] Andrew S Cassidy, Paul Merolla, John V Arthur, Steve K Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta, Jun Sawada, Theodore M Wong, Vitaly Feldman, Arnon Amir, Daniel Ben-Dayan Rubin, Filipp Akopyan, Emmett McQuinn, William P Risk, and Dharmendra S Modha. Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores. *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN 2013)*, 2013.

[6] Steve K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, Shyamal Chandra, Nicola Basilico, Stefano Carpin, Tom Zimmerman, Frank Zee, Rodrigo Alvarez-Icaza, Jeffrey A Kusnitz, Theodore M Wong, William P Risk, Emmett Mcquinn, Tapan K Nayak, Raghavendra Singh, and Dharmendra S Modha. Cognitive Computing Systems: Algorithms and Applications for Networks of Neurosynaptic Cores. *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN 2013)*, 2013.

[7] William Daniel Hillis. The Connection Machine. *Dissertation*, 1985.

[8] Giacomo Indiveri and Shih-chii Liu Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, X(X):1–17, 2015.

[9] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[10] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S Modha. A Digital Neurosynaptic Core Using Embedded Crossbar Memory with 45pJ per Spike in 45nm. *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, 2011.

[11] Paul A Merolla. communication network and interface A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 668, 2014.

[12] G E Moore. Cramming more components onto integrated circuits (Reprinted from Electronics, pg 114-117, April 19, 1965). *Proceedings Of The Ieee*, 86(1):82–85, 1998.

[13] John von Neumann. *First Draft of a Report on the EDVAC*. Texts and Monographs in Computer Science. Springer Berlin Heidelberg, 1982.