Neuromorphic Kalman filter implementation in IBM's TrueNorth

PRIME
PACIFIC RIM MEETING
ON ELECTROCHEMICAL AND SOLID STATE SCIENCE
HONOLULU, HI
Oct 6–11, 2024
Abstract submission deadline:
April 12, 2024
Learn more and submit!
Joint Meeting of
The Electrochemical Society
The Electrochemical Society of Japan
Korea Electrochemical Society

Neuromorphic Kalman filter implementation in IBM's TrueNorth

R. Carney, K. Bouchard, P. Calafiura, D. Clark, D. Donofrio, M. Garcia-Sciveres, J. Livezey

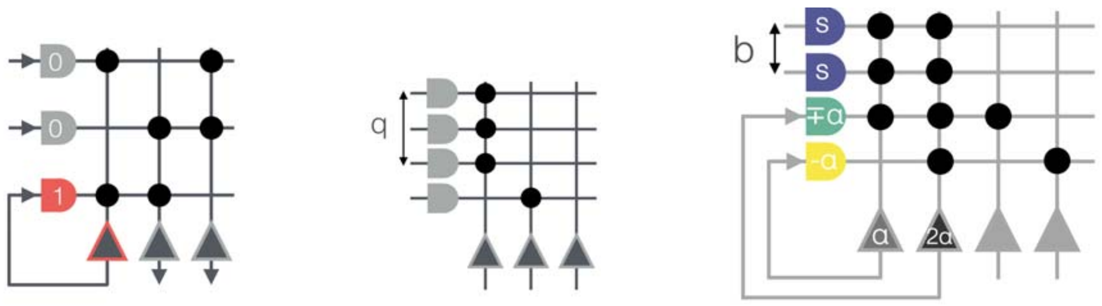
Lawrence Berkeley National Laboratory, USA

E-mail: `rcarney@lbl.gov`

Abstract. Following the advent of a post-Moore's law field of computation, novel architectures continue to emerge. With composite, multi-million connection neuromorphic chips like IBM's TrueNorth, neural engineering has now become a feasible technology in this novel computing paradigm. High Energy Physics experiments are continuously exploring new methods of computation and data handling, including neuromorphic, to support the growing challenges of the field and be prepared for future commodity computing trends. This work details the first instance of a Kalman filter implementation in IBM's neuromorphic architecture, TrueNorth, for both parallel and serial spike trains. The implementation is tested on multiple simulated systems and its performance is evaluated with respect to an equivalent non-spiking Kalman filter. The limits of the implementation are explored whilst varying the size of weight and threshold registers, the number of spikes used to encode a state, size of neuron block for spatial encoding, and neuron potential reset schemes.

1. Introduction

As High Energy Physics (HEP) experiments extend the range of attainable luminosities to produce more particle tracks per bunch crossing than ever before, reconstructing the tracks produced in detectors from such interactions becomes more challenging and new methods of computation and data-handling are being explored [1]. Additionally, understanding portability of HEP algorithms to future commodity computing architectures is necessary to project future computing costs. A key algorithm in track reconstruction in multiple HEP experiments over the past 50 years is the Kalman filter [2][3]. Implementing this algorithm in a neuromorphic architecture [4] represents a first step in understanding the benefits and limitation of introducing such a device into the computational resources available to HEP experiments. Neuromorphic chips are able to run computations at extremely low power, with respect to conventional computers, by encoding data in spikes instead of bits and using brain-inspired computational elements to process data. Although low power is not a major consideration for HEP tracking, this feature could be exploited to run neuromorphic technology at speeds more suited to data processing in HEP. This study aims to understand the feasibility of implementing a version of the Kalman filter algorithm in a neuromorphic architecture and to quantify how well the algorithm performs in a spiking implementation when compared to a non-spiking, conventional implementation.



(a) A basic crossbar. The axon numbers show different labels corresponding to 2 of 4 possible neuron weights.

(b) An input can be duplicated over q axons to increase the range of the neuron weight.

(c) The parallel multiplier xbar with blocksize 2. The third axon has a negative weight for neuron 1 and a positive weight for neuron 2.

Figure 1: TrueNorth core showing the inputs to the crossbar (axons - semicircles, left), internal connections (synapses - dots), and processing units (neurons - triangles, bottom).

2. TrueNorth

The architecture on which this study was deployed is the first instance of IBM Research's fully digital neuromorphic chip: TrueNorth [5]. The chip communicates and processes data packets called spikes. Spikes enter one of the 4096 cores in the chip through inputs called axons and are processed in computational elements inside each core called neurons. Each core can be connected to any other core on the chip by a two-dimensional mesh network, and therefore from each neuron on the crossbar to any axon on the chip. Axons can have one of four possible labels, so a spike that enters the crossbar through an axon of type 0 will be handled differently to a spike that enters through an axon of type 1, even if they are connected to the same neuron. The axons are connected to the neurons via binary connections, or synapses, in the crossbar, as is illustrated in Fig. 1a. TrueNorth cores are used only as needed and the communication of spikes happens asynchronously. However, there is a constraint that all spikes be processed in the relevant neuron, sent to their destination axon, and be queued in that axon, within a specified period called a tick - a synchronization point that all components within a core and all cores across the chip must adhere to [6]. The neuron model employed in TrueNorth is a digital variation on Leaky-Integrate-and-Fire [7] and executes the following five operations sequentially: neuron membrane potential update, leak update, check potential against threshold, and, following the result, either emit a spike and reset the neuron potential according to a given reset scheme or maintain the current neuron potential and wait for the next spike. The leak, in addition to neuron parameters such as the negative threshold, is not used in this implementation and shall not be discussed further. The complete neuron description can be found in [7]. The neuron membrane potential is updated at each tick by the addition of incoming, weighted spikes described by equation 1:

$$V_j(t) \;=\; V_j(t-1) \;+\; \sum_{i=1}^{256} n_i(t) \times W_{i,j} \times S_j^{G_i} \tag{1}$$

where $V_j(t)$ denotes the membrane potential associated with neuron j at tick t, $n_i(t) \in \{0,1\}$ indicates whether a spike was incident on axon i in the current tick, and $W_{ij} \in \{0,1\}$ indicates if neuron j and axon i are connected or not. The weight applied to the incoming spike is denoted by $S_j^{G_i}$, where G_i is the axon label with $i \in \{1,2,3,4\}$. TrueNorth programmers decide which of the synapses on a crossbar are connected, what label a particular axon has, the value of each of the four weights stored in each neuron, and, if multiple cores are used, how they are connected.

3. Review of Kalman filter notation

This work implements a steady-state Kalman filter [8] in the TrueNorth architecture. The steady-state Kalman filter tracks a linear dynamical system, as described in equation 2:

$$\vec{x}_{t+1} = \mathbf{F}\vec{x}_t + \vec{w}_t \ , \quad \vec{y}_t = \vec{x}_t + \vec{v}_t \tag{2}$$

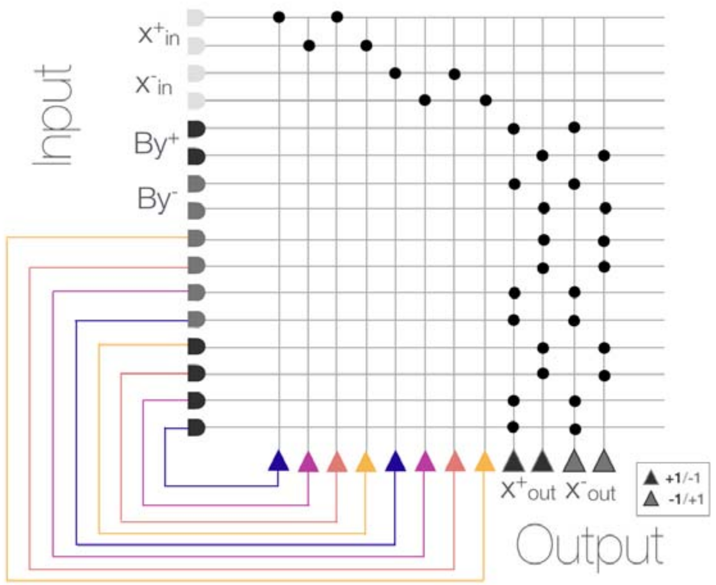
where $\vec{x}_t$ is an n-dimensional state vector at time t, $\mathbf{F}$ is the state transition matrix that encodes the dynamics of the system, $\vec{y}_t$ is the observed or measured output of the system, $\vec{w}_t$ is the process noise, and $\vec{v}_t$ is the measurement noise. The Kalman filter update is then given by equation 3:

$$\begin{aligned} \hat{x}_{t+1|t} &= (\mathbf{F} - \mathbf{L})\,\hat{x}_{t|t-1} + \mathbf{L}\vec{y}_t \\ &= \mathbf{A}\hat{x}_{t|t-1} + \mathbf{B}\vec{y}_t \end{aligned} \tag{3}$$

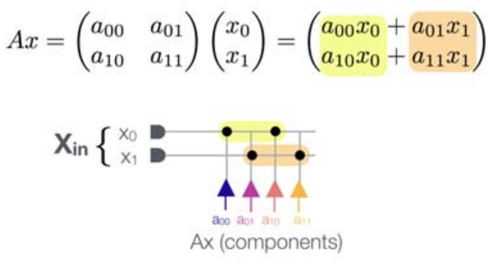
where $\hat{x}$ denotes the state estimate. In this work $\mathbf{A}$ is defined as the matrix multiplying the recurrent state estimate and $\mathbf{B}$ is the matrix multiplying the observations of the system, in this case $\mathbf{B}$ is equal to the Kalman gain, $\mathbf{L}$. Since steady-state dynamics are assumed, both the $\mathbf{A}$ and $\mathbf{B}$ matrices can be pre-calculated, or trained, before deploying the model in TrueNorth.

4. Multiplication and addition in TrueNorth

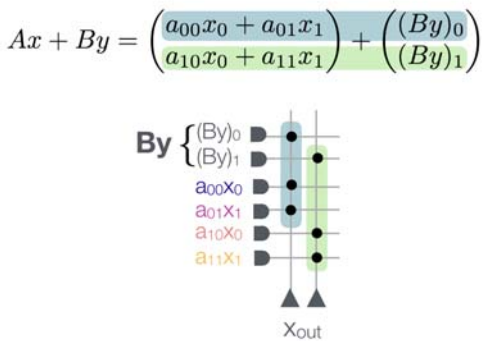
The hardware description language used by TrueNorth is called corelets, which describes the crossbar layout, neuron and axon parameter assignments, and inter-core connections. The



(a) Crossbar for 2-dim. Kalman filter corelet.


$Ax = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} a_{00}x_0 + a_{01}x_1 \\ a_{10}x_0 + a_{11}x_1 \end{pmatrix}$
Xin
X0
X1
a00 a01 a10 a11
Ax (components)

(b) Matrix multiplication.


$Ax + By = \begin{pmatrix} a_{00}x_0 + a_{01}x_1 \\ a_{10}x_0 + a_{11}x_1 \end{pmatrix} + \begin{pmatrix} (By)_0 \\ (By)_1 \end{pmatrix}$
By
(By)0
(By)1
a00X0
a01X1
a10X0
a11X1
Xout

(c) Linear algebra addition.

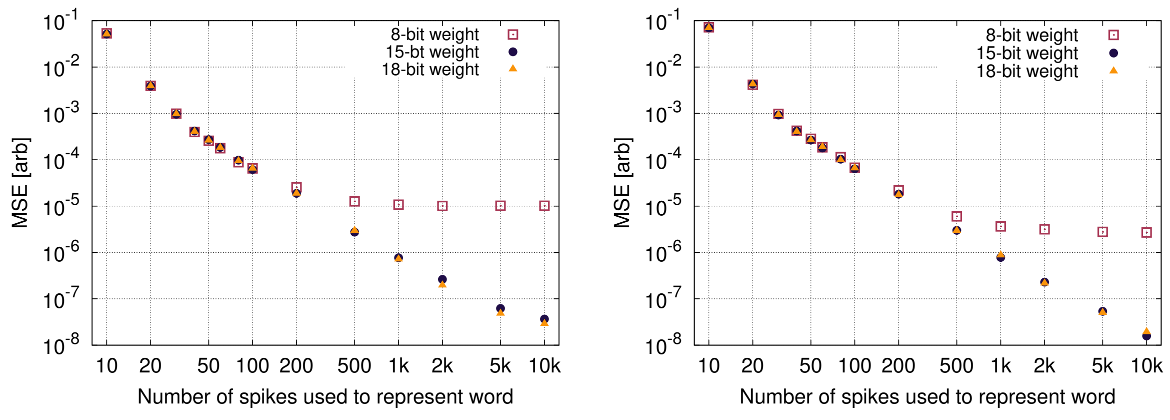
Figure 2: The crossbar for a two-state, serial Kalman filter. Positive and negative values are handled in parallel, as noted with the +/- indices. Axon labels are denoted by color, a key for which is shown in the legend next to the output neurons: axon type 0 multiplies the positive state estimate by +1, axon type 1 by -1 and the inverse for the negative state. Please note that this cartoon does not include the delay corelet that aligns the timing in the feedback circuit nor the recurrent logic to handle the positive and negative lines.

Kalman filter is designed from a set of corelet concepts: a multiplication corelet to multiply the previous state estimate matrix **A**, a delay corelet that aligns the timing in the feedback circuit, and an addition corelet that combines the latest noisy observation of the system with the delayed state estimate. Multiplication is implemented by changing the number of spikes coming into a multiplication corelet by a factor, a_{ij}, an element of the weight matrix **A**. The multiplicative factor is realised by changing the associated neuron threshold and synaptic weight such that the weight divided by the threshold is equal to a_{ij}. Addition is realised by connecting inputs in the same column of the crossbar. One of our implementations of the Kalman filter crossbar (labelled as serial) is shown in Fig. 2a for a 2-dim. system, with the multiplication and addition sections highlighted in Fig. 2b and 2c, respectively. The delay corelet (not shown in the figure) is implemented with a standard diagonal crossbar. Also not shown in this cartoon is the recurrent logic to combine the positive and negative addition channels.

5. Implementation and discussion

The Kalman filter was implemented in TrueNorth with the aim of maximising the accuracy of the state estimate with respect to its non-spiking counterpart, a steady-state Kalman filter implemented in Python. This section details how design choices affect that accuracy which is quantified by the the Mean Square Error (MSE) or variance of residuals when comparing the spiking and non-spiking implementations. Two versions of the Kalman filter were designed: a serial version which minimises the total number of cores used and a parallel version which minimises the latency of the system but uses more cores than the serial version. This section will also explore the unique considerations for this parallel design. Two models were tested on the TrueNorth Kalman filter: a 3-dim. state model encoding the position, velocity, and acceleration of a 1D projectile and a 2-dim. state model of the amplitude and phase of a 1D sine wave. The results of testing these models will be presented.

5.1. Input representation precision



(a) Encoding word scan for 3-dim. KF tracking a projectile. (b) Encoding word scan for 2-dim. KF tracking a sine wave.

Figure 3: The effect of the encoding word size on state estimation accuracy as compared to a non-spiking, ideal Kalman filter. A word is represented by a fixed number of spikes encoded in time, or, time and population.

Unary spikes are used to encode information in TrueNorth which allows for a number of options to encode data. In the case of the serial Kalman filter, rate encoding is used such that the value of a data word is given by the total number of spikes collected during the encoding window. The ordering of the spikes in this window is not important. In the parallel implementation the data word is encoded in two dimensions: an encoding window that lasts some number of ticks and a block which spans a number of axons:

$$\text{data word size} = \text{window of time [ticks]} \times \text{blocksize [axons]} \tag{4}$$

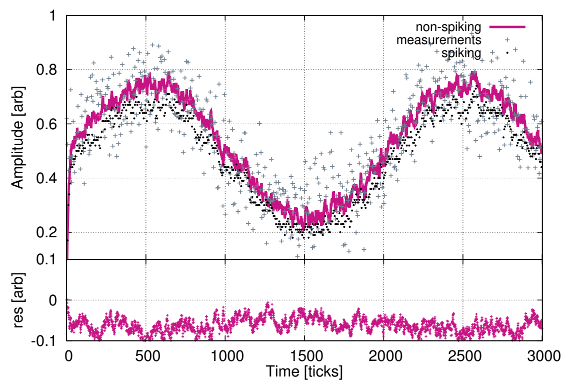
The number of spikes that can be used to represent a data word has a significant effect on the accuracy of the state estimate when compared with a non-spiking, steady-state Kalman filter, as can be seen in Fig. 3. Whilst both plots show increasing accuracy in predicting the state estimate for an increasing word size, the plots also show that the size of neuron weight register is important. Both the 3-dim. and 2-dim. models tested show that the accuracy is limited by using the default neuron weight register size of 8-bits. In both cases there is little difference in using a 15-bit weight (the maximum achievable using the duplicate axon method detailed in the following section) and an 18-bit weight (the size of the threshold register), but optimal size, beyond which there is a negligible improvement in prediction accuracy, is likely dependent on the weights being represented and hence the dynamics being tracked and is therefore model dependent.

5.2. Reset mode

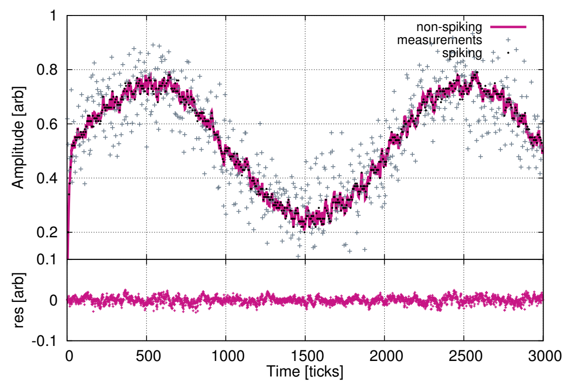
A reset mode describes what happens to V_j after that neuron has fired. The serial Kalman filter implementation simply uses the linear neuron model [7], where after a neuron fires its V_j reduces by the value of the threshold, α. To have the same behaviour across a block of neurons in the parallel implementation however, the potential in each block of neurons is reset by an amount proportional to the number of spikes that have been fired after a tick by the *entire* block - and each neuron in the block is reset by the same amount. This is achieved by a combination of the crossbar design and the linear neuron. The crossbar for this custom reset scheme can then be divided into two parts: a 'full' crossbar, where every input axon is connected to every neuron, and a 'reset' crossbar, which reduces the value of V_j in each neuron correctly, based on which neurons in the block have fired, for example see Fig. 1c for a crossbar of blocksize $= 2$. 'Population-encoding' has been done before in TrueNorth and this design shares the feature of a linear ladder of neuron thresholds that progress along each block with the design shown in [9] but what makes this implementation unique is that it preserves V_j remainders that cross from one data word to the next. A remainder in V_j occurs when the potential is not an integer multiple of the threshold such that, after spiking, V_j will be non-zero. Other algorithms used in TrueNorth either throw this remainder away after every tick, or reset V_j at the end of each data word. The effect of using this latter reset scheme on the accuracy of the serial Kalman filter is shown in Fig. 4. By losing the multiplication remainder from the previous word, the state estimate starts off consistently lower than it should, as can be seen when comparing Fig. 4a and 4b. This effect is mediated by increasing the size of the data word as can be seen in Fig. 4c and 4d. However, even with a large data word it is clear that for the TrueNorth Kalman filter, resetting V_j after every data word provides poorer accuracy for a similar crossbar complexity, thus corelet designs employing it were not used in this study.

5.3. Weight register precision

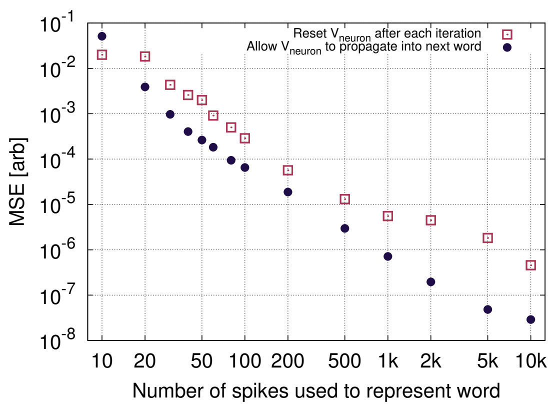
The Kalman filter uses a matrix of weights to multiply the recurrent connection. An element of this matrix is represented by the ratio of the neuron weight, s, and threshold, α. The neuron weights are stored in 8-bit-plus-sign registers $s \in \{-(2^8-1), 2^8-1\}$ and neuron thresholds in 18-bit registers, $\alpha \in \{0, 2^{18}-1\}$. A scheme, used in both the serial and parallel implementations, to increase the effective number of bits in the neuron weight register is to duplicate an incoming



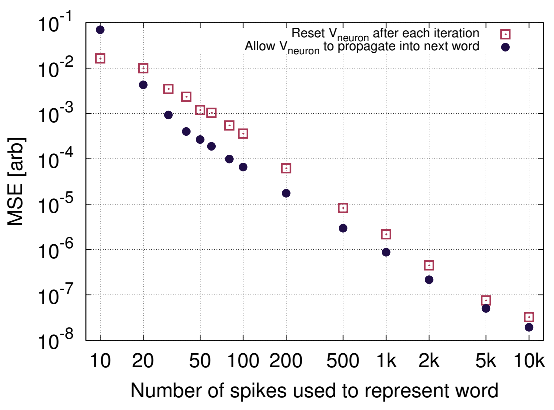
(a) V_j set to zero at the end of each data word, any remainder is discarded.



(b) V_j remainder allowed to propagate into next data word.



(c) Comparison of resetting V_j after each encoding window or not for a 3-dim. tracking problem.



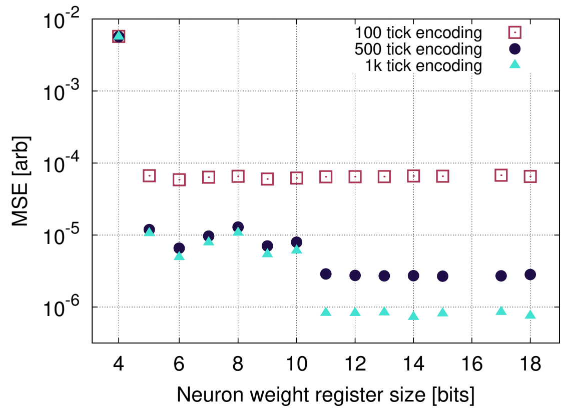
(d) Comparison of resetting V_j after each encoding window or not for a 2-dim. tracking problem.

Figure 4: Plots 4a and 4b show the effect of resetting V_j every sample on the spiking Kalman filter state estimate - it is particularly evident in plot 4a that resetting at the end of each data word causes the spiking Kalman filter to underestimate the state. Plots 4c and 4d compare the effect of the reset scheme over different encoding schemes for a 3-dim. and 2-dim. system, respectively.

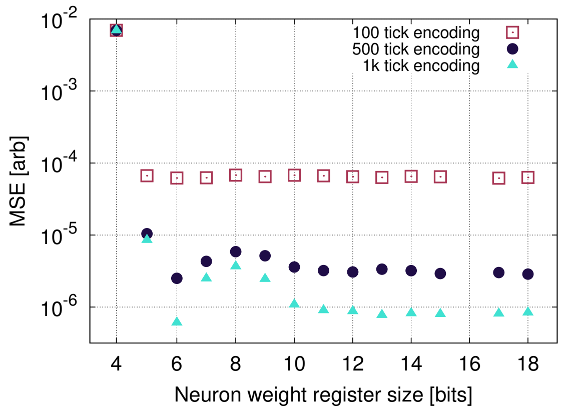
spike train over multiple axons connected to the same neuron (see Fig. 1b). This effectively increases the maximum value that can be represented by the neuron weights by $255 \cdot q$, where q is the number of axons the spike train is duplicated over. The effect of using different effective weight register sizes on the accuracy of a state estimate, with respect to the non-spiking equivalent, can be seen in Fig. 5. As with the size of the data word used, the optimal effective precision of the neuron weight register will be dependent on the encoding scheme used and the model the system tracks. However, there is clearly significant value in using axons on a core to increase the effective size of a neuron weight register but it is worth noting that there is a point below which the changes to the precision of the weight representation no longer change the accuracy of the state estimate for a given encoding scheme.

5.4. Physical limitations: core size, tick rate

The parameters discussed in previous sections have the flexibility to be extended beyond their initial values in TrueNorth but a feature that cannot be changed is the size of a core and this



(a) Weight register size scan for a 3-dim Kalman filter tracking a projectile.



(b) Weight register size scan for a 2-dim Kalman filter tracking a sine wave.

Figure 5: The effect of the neuron weight register size on state estimation accuracy as compared to a non-spiking, ideal Kalman filter.

affects the accuracy that can be achieved for a given latency. Matrix multiplication in the parallel implementation of the Kalman filter cannot be done in a single core (unlike for the serial implementation as was seen in Fig. 2a). Instead, each core encodes an individual element in the multiplication matrix for maximum accuracy and reduced latency and as such the plot in Fig. 6 is independent of the size of the state vector. However, the state vector size is limited in the parallel implementation by the addition corelet, both by the core size and by the size of the state vector, in the following relation:

$$(2n+1)\,b \;\leq\; 256 \tag{5}$$

where n is the size of the state-vector, b is the blocksize, and 256 is the number of axons in a core. This equation puts an upper limit on the blocksize for a given state-vector but does not directly change the accuracy of the state estimate - that occurs in the multiplication corelet. Because of the finite size of both the neuron weight register (8-bit) and the limited number of axon labels (4), the parallel implementation requires doubling the number of neurons for each input axon to correctly reset the parallel block. This limits the maximum blocksize to $b = 21$. In the serial implementation the number of times axons can be duplicated to achieve a more precise representation of the real weights is limited exclusively by the size of the core:

$$\sum_{i=1}^{n} q_i n_i \;\leq\; 256 \tag{6}$$

where n is the size of the state vector and q is the number of repetitions to increase weight range, assuming the entire state is processed in one core. However, in the parallel implementation:

$$p\left(\frac{b(b+1)}{2}-1\right)+bq \;\leq\; 256 \tag{7}$$

where p is the number of repetitions needed to increase the threshold range. Fig. 6 shows, for an encoding of 1000 ticks (which is the serial case), the maximum effective number of bits in the neuron weight when using the parallel implementation for a given blocksize. A larger blocksize leaves less physical space on the core for axon duplication which results in the steps of equal precision for varying blocksize shown in the plot. The plot does show that for the limited size of the crossbar on TrueNorth, a compromise between system latency and estimate accuracy must be decided for the parallel implementation.

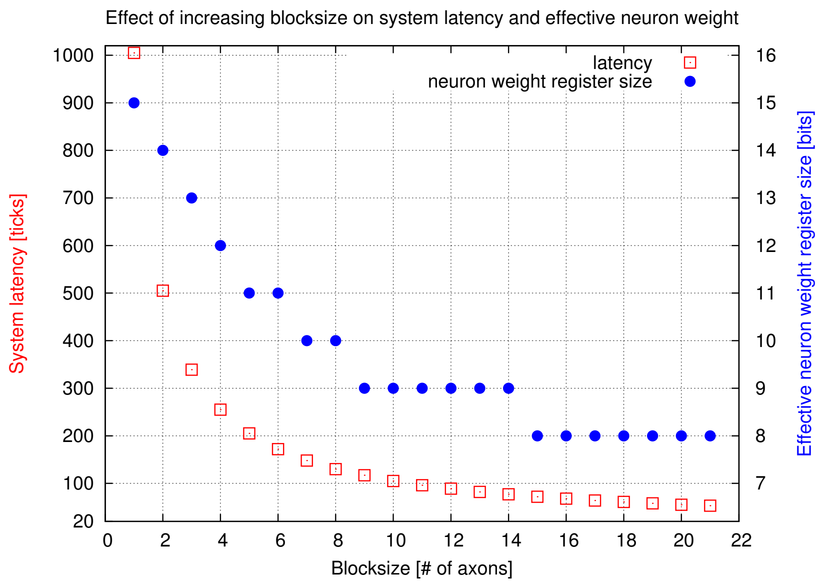


Figure 6: The effect of increasing the blocksize, or the number of axons that encode the data in parallel, on the latency of the system and the effective size of the neuron weight register for a 1000 tick encoding scheme.

6. Conclusion

This work is the first implementation of a Kalman filter in the TrueNorth architecture and shows that for the models tested the spiking implementation performs similarly to the non-spiking implementation. The number of spikes used to encode the state and the neuron weight register size both effect this accuracy and are interdependent. A serial and parallel implementation of the Kalman filter have been developed to trade-off the number of cores versus the latency of output. It was found, when using the parallel implementation, that reducing the latency decreases the effective weight register size, and hence accuracy of the state estimates it produces.
This work presents a promising first step in understanding how HEP tracking algorithms could be ported to neuromorphic computing. In the second year of this project an implementation of a neuromorphic platform will be developed in FPGA's to study architectural trade-off's beyond the constraints of TrueNorth. The TrueNorth chip development itself will be taken over by a neuroscience team for brain-machine interface applications.

References

[1] Salzburger A 2015 *Proc. CHEP 2015, J. Phys.: Conf. Ser.* **664** 072042
[2] Fruhwirth R 1987 *Nucl. Instrum. Meth.* **A262** 444-450
[3] Cerati G, et. al. 2016 *Proc. CTD, EPJ Web Conf.* **127** 00010
[4] Schuller I K, et. al, 2015 *DOE report: Neuromorphic Computing*
[5] Merolla P A, et. al. 2014 *Science* **345** 6197 (668-673)
[6] Akopyan F, et. al. 2015 *IEEE Transactions on CAD of Integrated Circuits and Systems* **34** 10 (1537-1557)
[7] Cassidy A, et. al. 2013 *Proc. International Joint Conf. on Neural Networks* IEEE
[8] Kailath T 1981 *Lectures on Wiener and Kalman Filtering* Springer Vienna (110-114)
[9] Andreopoulos A, et. al 2016 *Proc. International Joint Conf. on Neural Networks* IEEE