frontiers in
NEUROSCIENCE

ORIGINAL RESEARCH ARTICLE
published: 22 January 2015
doi: 10.3389/fnins.2015.00002

Reducing the computational footprint for real-time BCPNN learning

Bernhard Vogginger[1]*, René Schüffny[1], Anders Lansner[2,3], Love Cederström[1], Johannes Partzsch[1] and Sebastian Höppner[1]

[1] *Department of Electrical Engineering and Information Technology, Technische Universität Dresden, Germany*
[2] *Department of Computational Biology, School of Computer Science and Communication, Royal Institute of Technology (KTH), Stockholm, Sweden*
[3] *Department of Numerical Analysis and Computer Science, Stockholm University, Stockholm, Sweden*

Edited by:
Sadique Sheik, University of Zurich and ETH Zurich, Switzerland

Reviewed by:
Guillaume Garreau, Johns Hopkins University, USA
Sergio Davies, The University of Manchester, UK
Srinjoy Das, University of California, San Diego, USA

****Correspondence:***
Bernhard Vogginger, Chair for Highly Parallel VLSI Systems and Neuromorphic Circuits, Institue of Circuits and Systems, Department of Electrical Engineering and Information Technology, Technische Universität Dresden, 01062 Dresden, Germany
e-mail: bernhard.vogginger@tu-dresden.de

The implementation of synaptic plasticity in neural simulation or neuromorphic hardware is usually very resource-intensive, often requiring a compromise between efficiency and flexibility. A versatile, but computationally-expensive plasticity mechanism is provided by the Bayesian Confidence Propagation Neural Network (BCPNN) paradigm. Building upon Bayesian statistics, and having clear links to biological plasticity processes, the BCPNN learning rule has been applied in many fields, ranging from data classification, associative memory, reward-based learning, probabilistic inference to cortical attractor memory networks. In the spike-based version of this learning rule the pre-, postsynaptic and coincident activity is traced in three low-pass-filtering stages, requiring a total of eight state variables, whose dynamics are typically simulated with the fixed step size Euler method. We derive analytic solutions allowing an efficient event-driven implementation of this learning rule. Further speedup is achieved by first rewriting the model which reduces the number of basic arithmetic operations per update to one half, and second by using look-up tables for the frequently calculated exponential decay. Ultimately, in a typical use case, the simulation using our approach is more than one order of magnitude faster than with the fixed step size Euler method. Aiming for a small memory footprint per BCPNN synapse, we also evaluate the use of fixed-point numbers for the state variables, and assess the number of bits required to achieve same or better accuracy than with the conventional explicit Euler method. All of this will allow a real-time simulation of a reduced cortex model based on BCPNN in high performance computing. More important, with the analytic solution at hand and due to the reduced memory bandwidth, the learning rule can be efficiently implemented in dedicated or existing digital neuromorphic hardware.

Keywords: Bayesian confidence propagation neural network (BCPNN), Hebbian learning, synaptic plasticity, event-driven simulation, spiking neural networks, look-up tables, fixed-point accuracy, digital neuromorphic hardware

1. INTRODUCTION

Bayesian Confidence Propagation Neural Networks (BCPNNs) realize Bayesian statistics with spiking or non-spiking neural networks. They can be used to build powerful associative memories (Sandberg et al., 2000; Meli and Lansner, 2013) and data classifiers, with applications ranging from data mining (Bate et al., 1998; Lindquist et al., 2000) to olfaction modeling (Kaplan and Lansner, 2014). The underlying Bayesian learning rule has clear links to biological synaptic plasticity processes (Tully et al., 2014), cortical associative memory (Lansner, 2009), reinforcement learning (Johansson et al., 2003), and action selection (Berthet et al., 2012). Furthermore, BCPNNs have been used to model phenomena like synaptic working memory (Sandberg et al., 2003), word-list learning in humans (Lansner et al., 2013) and memory consolidation (Fiebig and Lansner, 2014), making it a promising paradigm for information processing in the brain, while retaining a level of abstraction suitable for efficient technical implementation. Models using more detailed spiking attractor networks with the same structure have provided non-trivial explanations for memory retrieval and other basic cognitive phenomena like e.g., attentional blink (Lundqvist et al., 2010, 2011; Silverstein and Lansner, 2011; Lundqvist et al., 2013).

The performance of BCPNNs, for example in memory tasks, scales well with network size, making them extraordinarily powerful for large networks (Johansson et al., 2001). Therefore, massively parallel simulations of these networks (29 million spiking units, 295 billion plastic connections) have been realized on supercomputers (Benjaminsson and Lansner, 2011). These showed that BCPNN implementations are bounded by computation (Johansson and Lansner, 2007). To alleviate this limit, conceptual work on implementations in neuromorphic hardware has been performed (Johansson and Lansner, 2004; Farahini et al., 2014; Lansner et al., 2014).

In this paper, we pave the way for an efficient implementation of BCPNN in digital neuromorphic hardware by reducing both its computational and memory footprint. Existing software models

apply fixed step size numerical integration methods for solving the BCPNN dynamics. Although easy to implement, this clock-driven simulation approach has two major drawbacks: First, there is a relatively high base cost for calculating the updates of all state variables at every time step, irrespective of the spiking activity in the network. Second, the states have to be read from and written back to memory at every simulation step, which is especially expensive for custom hardware implementations where the states are stored in an external memory. As suggested in recent work (Lansner et al., 2014), we tackle these issues by moving to an event-driven simulation scheme, which we systematically optimize for minimal number of calculations to achieve a reduction of the computational load by an order of magnitude. This efficiency gain of the event-driven paradigm is mainly due to the sparse activity in BCPNNs, which is retained irrespective of network size. Employing pre-calculated look-up tables for the frequent calculation of the exponential function, we further minimize the computational cost per event-driven update. By using an analytical solution of the model equations, the numerical accuracy of the simulation is increased compared to conventional simulation techniques with fixed step size (Henker et al., 2012). We show how this accuracy overhead could be utilized for significantly reducing the required memory and memory bandwidth in a potential hardware implementation by using fixed point operands with fewer bits than in a floating point representation.

While we performed our equation optimizations specifically for the BCPNN model, they are not restricted to it. As BCPNNs rely on dynamic equations that are common in neuroscientific modeling, our approach can be easily adopted to other models. It shows how to efficiently calculate single neuronal traces and correlation measures for synaptic plasticity, increasing the energy efficiency of digital implementations, either on standard computers or on specialized hardware, on an algorithmic level, complementing analog approaches for increasing the energy efficiency of neuromorphic computation (Hasler and Marr, 2013).

2. MATERIALS AND METHODS

2.1. BAYESIAN CONFIDENCE PROPAGATION NEURAL NETWORKS

In BCPNNs (Lansner and Ekeberg, 1989; Lansner and Holst, 1996) the synaptic weights between network units are calculated in a Hebbian fashion by applying Bayes' rule on the past activity of the units giving a measure of the co-activation of the units. In a similar manner each unit's bias is calculated from its past activity, representing its *a priori* probability to be active. Often, the activity of the units is represented by stochastic spike events, which are generated according to each unit's recent input and own activity. Typically, in a *training* phase these correlation and activation statistics are collected, which are then used in the subsequent *test* phase to perform inference, i.e., to determine the *a posteriori* activity of some units as a response to other units' recent activity. While the concept of BCPNN was originally developed for series of discrete samples, a time-continuous spike-based version has been developed recently, which we describe in Section 2.1.1 and whose efficient simulation is the main subject of this article. In Section 2.1.2, we present an application of this spike-based BCPNN learning rule in a modular network that constitutes a reduced full-scale model of the cortex.

2.1.1. Spike-based BCPNN

Spike-based BCPNN (Wahlgren and Lansner, 2001; Tully et al., 2014) is implemented by a set of local synaptic state variables that keep track of presynaptic, postsynaptic, and synaptic (i.e., correlated) activity over three different time scales, by passing spiking activity over three low pass filters, see **Figure 1**. Here and throughout this paper the three sites (pre-, postsynaptic and synaptic) are denoted by indices i, j, and ij, respectively. In the first processing stage, the pre- and postsynaptic spiking activity represented by spike trains S_i (resp. S_j) is low pass filtered into the Z_i and Z_j traces (**Figure 1B**), with time constants τ_{z_i} and τ_{z_j} in a range of 5 ms to 100 ms, which corresponds to typical synaptic decay time constants for various receptor types.

In the second stage, the Z traces are passed on to the E or eligibility traces and low pass filtered with time constant τ_e. Here, a separate trace E_{ij} is introduced to filter the coincident activity of the Z-traces, see **Figure 1C**. The E traces typically have slower dynamics than the Z traces ($\tau_e \approx 20 - 1000$ ms), and can be motivated to provide a mechanism for delayed reward learning (cf. Tully et al., 2014).

The E traces in turn are low pass filtered into the P traces (**Figure 1D**). These tertiary traces have the slowest dynamics with time constant τ_p ranging from 1 s to several 100 s, even higher values are possible. The P traces correspond to the probabilities of the units being active or co-active in the original non-spiking BCPNN formulation (Lansner and Holst, 1996). In a final step the P traces are used to compute the synaptic weight w_{ij} and the postsynaptic bias β_j (**Figure 1E**). The formulas for w_{ij} and β_j contain the parameter ϵ, which originates from a minimum spiking activity assumed for the pre- and postsynaptic units (cf. Tully et al., 2014), and which has the side effect to avoid division by zero in the weight formula.

The global parameter κ in the dynamics of P traces can take any non-negative value and controls the learning, i.e., it determines how strong recent correlations are stored. When the learning rate κ equals zero, there is no learning, as the P traces do not change at all, and thus neither do the synaptic weight w_{ij} and the postsynaptic bias β_j. We assume that κ only undergoes discrete and seldom changes, mostly when learning is switched on or off. Hence, while κ is constant and non-zero, the dynamics of the P traces can be expressed with a modified time constant τ_p^*:

$$\tau_p^* \frac{dP}{dt} = E - P, \quad \tau_p^* = \frac{\tau_p}{\kappa} \tag{1}$$

We refer to Tully et al. (2014) for establishing the link between the spike-based and the probabilistic BCPNN learning rule, as well as for details on the biological equivalents of the processing stages. Also, note that in some cases the second low pass filter is not actually used, so that the Z traces are directly passed to the P traces.

2.1.2. Reduced modular model of the cortex

As an application of the spike-based BCPNN we consider a modular abstract network model, motivated by the columnar structure of the cortex, that was already presented in Lansner et al. (2014). One assumption is that the smallest functional units

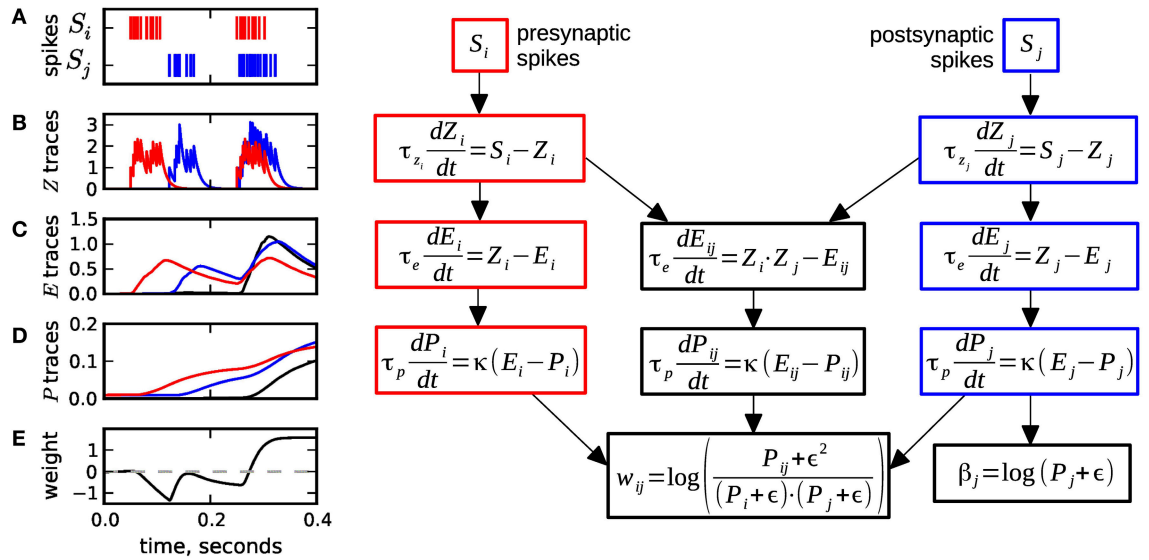

FIGURE 1 | Equations and sample traces of the spike-based BCPNN learning rule. (A) Presynaptic (red) S_i and postsynaptic (blue) S_j spike trains serve as input to a BCPNN synapse. **(B)** The input spike trains are low pass filtered into the Z traces with time constants τ_{z_i}, τ_{z_j}. **(C)** E traces compute the low-pass filter of Z traces with τ_e. The E_{ij} variable (black) tracks coincident pre- and postsynaptic activity. **(D)** E traces are passed on to the P traces and low-pass filtered with τ_p. **(E)** The P traces are used to compute the postsynaptic bias β_j and the synaptic weight w_{ij}, which can vary between positive and negative values. Usually, the dynamics get slower from **B** to **D**: $\tau_{z_i}, \tau_{z_j} \leq \tau_e \leq \tau_p$. Figure redrawn from Tully et al. (2014).

in the mammalian cortex are not single neurons but so-called minicolumns. A minicolumn is formed by a local population of some hundred neurons with enhanced recurrent connectivity and similar receptive fields, so that these neurons are assumed to have quite correlated output. An example would be a minicolumn encoding a certain orientation during processing in primary visual cortex.

In the order of 100 minicolumns are aggregated in a larger columnar structure, the cortical hypercolumn, which contains in the order of 10,000 neurons. Within a hypercolumn the minincolumns compete in a soft winner-take all (soft-WTA) fashion through feedback inhibition, so that most of the time only one minicolumn shows high firing activity while the others are mostly silent. Minicolumns can be viewed to encode a discrete value of an attribute specific to each hypercolumn.

In our reduced, abstract model, each minicolumn is represented by one stochastically spiking minicolumn unit (MCU). Only connections outside a hypercolumn are implemented: The internal connections between neurons of a minicolumn are hidden within the MCU, while the competitive feedback inhibition of the 100 MCUs within a hypercolumn unit (HCU) is hardwired by means of a normalization of activity per HCU (cf. Equation 5 below). In turn, for the implementation of the incoming long-range synaptic connections, which on the neuron level typically make up half of the between 10^3 and 10^4 incoming connections in total, we assume that each MCU propagates its spikes to 10,000 other MCUs, and has appropriately as many incoming connections. These connections are *patchy* in the sense that each MCU projects onto 100 hypercolumns and delivers spikes to all 100 MCUs of each target HCU. The connection scheme is motivated as follows: Long-range connections are provided by large layer 5 pyramidal cells, which make up around 10% of a minicolumn. Each of those cells forms synaptic connections to clusters of far away neurons in horizontal direction. The diameter of these clusters approximately corresponds to the dimension of a hypercolumn. In real cortex, each of the large pyramidal cells generates around 10 of these *patches* (Houzel et al., 1994; Binzegger et al., 2007), which motivates the 100 target HCUs per MCU, assuming that one MCU comprises one hundred neurons. All of these connections between MCUs are subject to the spike-based BCPNN learning equations of **Figure 1**.

At a higher level, HCUs represent independent network modules between which spikes are transmitted. Each HCU consists of 100 MCUs and 1 million plastic synapses organized in an array with 10^4 inputs and 100 outputs, as illustrated in **Figure 2**. The pre- and postsynaptic states of the BCPNN model can therefore be implemented at the margin of the array, while the synaptic traces E_{ij},P_{ij}, and w_{ij} form the array, thus representing the largest amount of state variables. The minicolumn units integrate the incoming spiking activity, which is then turned into a spiking probability of each unit. In particular, presynaptic input leads to a synaptic current $s_{\mathrm{syn,j}}$ (Equation 2), which together with the bias β_j and a specific external input I_j sums up to the support value s_j for each minicolum unit j in Equation (3):

$$\tau_{z_i}\frac{ds_{\mathrm{syn,j}}(t)}{dt} = \sum_i w_{ij}(t)S_i(t) - s_{\mathrm{syn,j}}(t) \qquad (2)$$

$$s_j(t) = \beta_j(t) + s_{\mathrm{syn,j}}(t) + I_j(t) \quad . \qquad (3)$$

The low-pass filtered version of Equation (3) gives the "membrane potential" m_j of each MCU:

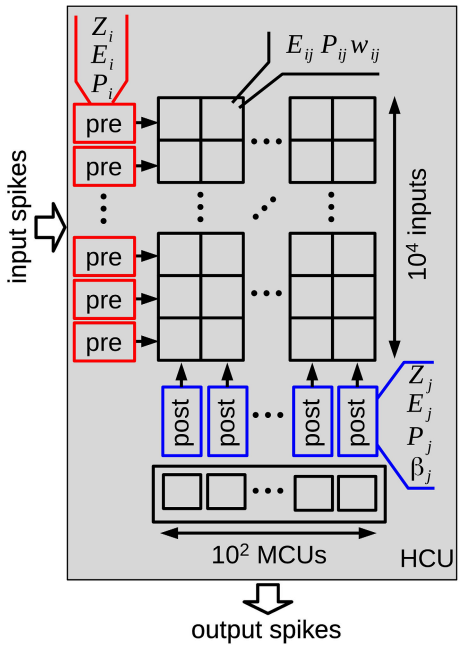

FIGURE 2 | Structure of one hypercolumn unit (HCU) of the reduced cortex model. Each HCU contains 100 stochastic minicolumn units (MCUs) that compete in a winner-take-all fashion. Input spikes from 10,000 MCUs of other HCUs connect via 1 million BCPNN synapses to all 100 MCUs of the HCU. Each MCU sends its output spikes to 100 other HCUs. In order to store all Z, E, P traces and synaptic weights of the HCU, more than 12 MB memory is required when using floating point numbers with single-precision.

$$\tau_m \frac{dm_j(t)}{dt} = s_j(t) - m_j(t) \quad , \tag{4}$$

where τ_m is the membrane time constant in the order of 10 ms. In other words, the MCUs are leaky-integrators (Equation 4) with three different input currents (Equation 3): The bias $\beta_j(t)$ represents the *prior* contribution to the unit's activation irrespective of the current synaptic input, determined by the past spiking activity of the unit itself via the postsynaptic traces (Z_j, E_j, P_j, cf. **Figure 1**). The synaptic input is implemented as an exponentially decaying synaptic current $s_{\mathrm{syn,j}}(t)$ (Equation 2), which - at a presynaptic spike of input i - is increased by synaptic weight $w_{ij}(t)$ learned according to the Equations in **Figure 1**. Last, the external input $I_j(t)$ allows a specific stimulation of single units.

All M MCUs of a hypercolumn unit are organized as a probabilistic soft-WTA circuit. The activation o_j of each unit is computed as:

$$o_j = \begin{cases} \dfrac{e^{\gamma_m m_j}}{\sum_{k=1}^{M} e^{\gamma_m m_k}}, & \text{if } \sum_{k=1}^{m} e^{\gamma_m m_k} > 1 \\ e^{\gamma_m m_j} & \text{otherwise} \end{cases} , \tag{5}$$

The gain factor γ_m controls the strength of the soft-WTA filtering process, the higher γ_m the higher the activation-ratio between the winning unit and the remaining units. The normalization in Equation (5) ensures that on average not more than 1 MCU is active at the same time.

The activation o_j then translates into the instantaneous Poisson firing rate r_j for each unit:

$$r_j(t) = o_j(t) \cdot r_{\max,\mathrm{HCU}} \tag{6}$$

where $r_{\max,\mathrm{HCU}}$ is the maximum firing rate per HCU. The average spiking frequency in mammalian cortex is quite sparse, with an average spike rate on the order of 0.1 Hz (Lennie, 2003). In our full scale HCU with 100 MCUs the average activity level would be around 1 Hz (thus $r_{\max,\mathrm{HCU}} = 100$ HZ), and the difference is explained by the fact that one MCU represents around 10 layer 5 pyramidal cells.

2.2. SIMULATION STRATEGIES

2.2.1. Fixed step size simulation

The typical approach for the simulation of spiking neural networks is simulation with fixed step size, where all states are synchronously updated at every tick of a clock (Brette et al., 2007; Henker et al., 2012). Usually, in such time-driven simulation, one uses numerical integration methods like Euler or Runge-Kutta to advance the state by one time step dt.

For our reference fixed step size simulation we follow Lansner et al. (2014) and use the explicit Euler method for the numerical integration with a rather long time step of dt =1 ms. As the MCUs are stochastic, the instantaneous firing rate r_j (Equation 6) is transformed into a firing probability per time step, which is then compared to a uniform random number between 0 and 1 to generate spikes. The 1 ms time step is also used in state-of-the-art real-time digital neuromorphic systems like the SpiNNaker (Furber et al., 2014) and the Synapse hardware (Merolla et al., 2014). For completeness, we also present results with 0.1 ms step size, which is commonly used for the simulation of spiking neural networks.

2.2.2. Event-driven simulation

In Sections 2.3.1 and 2.3.3 we provide analytical solutions for the spike-based BCPNN model. For those simulations we mix the time-driven and event-driven approach: We restrict spike times to multiples of the simulation time step dt. The stochastic MCUs (Equations 2–6) are evaluated as for the time-driven approach, which requires that also the β_j is computed at every time step. In contrast, the states of the BCPNN synapses (**Figure 1**) are only updated at the occurrence of a pre- or postsynaptic event.

2.3. ANALYTICAL SOLUTIONS OF SPIKE-BASED BCPNN

The simulation of spike-based BCPNN with a fixed step size method is cost-intensive and requires very frequent read and write of the state variables from and to memory. Therefore, we first provide the rather straightforward analytical solution of the BCPNN equations in Section 2.3.1, allowing an exact event-driven simulation scheme. As intermediate step, we rewrite the BCPNN dynamics as a spike response model in Section 2.3.2, which then provides the basis for a second analytical solution with reduced number of operations (Section 2.3.3). Although not

employed in the experiments in this article, discrete changes of the learning rate κ must be considered for the completeness of the two analytical solutions, which is done in Appendix A3 .

2.3.1. BCPNN solution: analytical I

For the event-driven simulation of BCPNN, the update of the state variables is only triggered by events (usually pre- or postsynaptic spikes). For each state variable one requires the time of its last update t^{last}, in contrast to the time-driven simulation, where all states correspond to the same global time. Event-driven simulations are especially efficient if the update of the states from t^{last} to the current time t can be solved analytically. Without further derivation, we give the analytic solution to advance the Z, E and P traces by $\Delta t = t - t^{\text{last}}$ from time t^{last} to t, provided that there is no spike between t^{last} and t. For the presynaptic traces the solutions are

$$Z_i(t) = Z_i(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_{z_i}}} + S_i(t) \tag{7}$$

$$E_i(t) = E_i(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_e}} + Z_i(t^{\text{last}}) a_i \left(\mathrm{e}^{-\frac{\Delta t}{\tau_{z_i}}} - \mathrm{e}^{-\frac{\Delta t}{\tau_e}} \right) \tag{8}$$

$$\begin{aligned} P_i(t) = {} & P_i(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} + a_i b_i \left(\mathrm{e}^{-\frac{\Delta t}{\tau_{z_i}}} - \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} \right) Z_i(t^{\text{last}}) \\ & + \left(E_i(t^{\text{last}}) - a_i Z_i(t^{\text{last}}) \right) c \left(\mathrm{e}^{-\frac{\Delta t}{\tau_e}} - \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} \right) \quad , \end{aligned} \tag{9}$$

with the following coefficients used for brevity:

$$a_i = \frac{\tau_{z_i}}{\tau_{z_i} - \tau_e} \quad , \quad b_i = \frac{\tau_{z_i}}{\tau_{z_i} - \tau_p^*} \quad , \quad c = \frac{\tau_e}{\tau_e - \tau_p^*} \quad . \tag{10}$$

In Equation (7) S_i describes the presynaptic spike train taking value 1 at the spike time t_i^f and value 0 otherwise, formally

$$S_i(t) = \textstyle\sum_{t_i^f} \delta(t - t_i^f) \quad , \tag{11}$$

where $\delta(\cdot)$ denotes a Dirac pulse. We note that Equation (8) is only valid when $\tau_{z_i} \neq \tau_e$, Equation (9) furthermore requires that τ_p^* is different from both τ_{z_i} and τ_e. For the sake of simplicity we restrict ourselves within this article to time constants fulfilling this condition, but give the solution for the other cases in Appendix A.

The update formulas for the postsynaptic traces Z_j, E_j, and P_j can be obtained by replacing indices i by j in the presynaptic update formulas.

Accordingly, the update of the synaptic traces E_{ij} and P_{ij} is given by:

$$E_{ij}(t) = E_{ij}(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_e}} + Z_i(t^{\text{last}}) Z_j(t^{\text{last}}) a_{ij} \left(\mathrm{e}^{-\frac{\Delta t}{\tau_{z_{ij}}}} - \mathrm{e}^{-\frac{\Delta t}{\tau_e}} \right) \tag{12}$$

$$\begin{aligned} P_{ij}(t) = {} & P_{ij}(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} + a_{ij} b_{ij} \left(\mathrm{e}^{-\frac{\Delta t}{\tau_{z_{ij}}}} - \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} \right) Z_i(t^{\text{last}}) Z_j(t^{\text{last}}) \\ & + \left(E_{ij}(t^{\text{last}}) - a_{ij} Z_i(t^{\text{last}}) Z_j(t^{\text{last}}) \right) c \left(\mathrm{e}^{-\frac{\Delta t}{\tau_e}} - \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} \right), \end{aligned} \tag{13}$$

with shortcuts

$$\tau_{z_{ij}} = \left(\frac{1}{\tau_{z_i}} + \frac{1}{\tau_{z_j}} \right)^{-1} \quad , a_{ij} = \frac{\tau_{z_{ij}}}{\tau_{z_{ij}} - \tau_e} \quad , b_{ij} = \frac{\tau_{z_{ij}}}{\tau_{z_{ij}} - \tau_p^*} \quad . \tag{14}$$

Note that, on purpose, Equations (9, 13) were not further simplified to ease the comparison with the spike response model formulation of the BCPNN model in the next section. Again, we restrict ourselves to parameter sets where none of the involved time constants ($\tau_{z_{ij}}$, τ_e and τ_p^*) are equal. Note, however, that τ_{z_i} and τ_{z_j} may be equal.

The analytical solution of the BCPNN equations derived in this section is henceforth denoted as *analytical I* method.

2.3.2. Spike response model formulation of the BCPNN model

As starting point for a second event-driven analytical solution with less operations, we make use of the linearity of the BCPNN differential equations and formulate the dynamics as a spike response model, in accordance with the work of Gerstner and Kistler (2002). The presynaptic traces can be written as a response to spike times t_i^f:

$$Z_i(t) = \sum_{t_i^f} \zeta_i(t - t_i^f), \quad \zeta_i(t) = \mathrm{e}^{-\frac{t}{\tau_{z_i}}} \Theta(t) \tag{15}$$

$$E_i(t) = \sum_{t_i^f} \alpha_i(t - t_i^f), \quad \alpha_i(t) = a_i \left(\mathrm{e}^{-\frac{t}{\tau_{z_i}}} - \mathrm{e}^{-\frac{t}{\tau_e}} \right) \Theta(t) \tag{16}$$

$$\begin{aligned} P_i(t) = \sum_{t_i^f} \pi_i(t - t_i^f), \quad \pi_i(t) = {} & a_i \left[b_i \left(\mathrm{e}^{-\frac{t}{\tau_{z_i}}} - \mathrm{e}^{-\frac{t}{\tau_p^*}} \right) \right. \\ & \left. + c \left(\mathrm{e}^{-\frac{t}{\tau_p^*}} - \mathrm{e}^{-\frac{t}{\tau_e}} \right) \right] \Theta(t). \end{aligned} \tag{17}$$

Here $\Theta(\cdot)$ denotes the Heaviside step function. ζ_i, α_i, and π_i are the spike response kernels for the Z_i, E_i and P_i traces. One can obtain Equations (15–17) from the analytical solution by setting $Z_i(t^{\text{last}}) = 1, E_i(t^{\text{last}}) = 0, P_i(t^{\text{last}}) = 0$ in Equations (7–9). The spike response kernels ζ_i, α_i, and π_i are shown in the left panel of **Figure 3** as dashed lines. The postsynaptic traces can be analogously formulated, by replacing i with j in Equations (15–17).

For the synaptic trace variables E_{ij} and P_{ij} the spike response formulation becomes more sophisticated: Therefore, we consider the product $Z_i Z_j$, which after inserting the spike response formulation of Z_i and Z_j is given by:

$$Z_i Z_j = \sum_{t_i^f} \zeta_i(t - t_i^f) \cdot \sum_{t_j^f} \zeta_j(t - t_j^f) \tag{18}$$

$$= \sum_{t_i^f} \sum_{t_j^f} \mathrm{e}^{-\frac{t - t_i^f}{\tau_{z_i}}} \mathrm{e}^{-\frac{t - t_j^f}{\tau_{z_j}}} \Theta(t - t_i^f) \Theta(t - t_j^f) \tag{19}$$

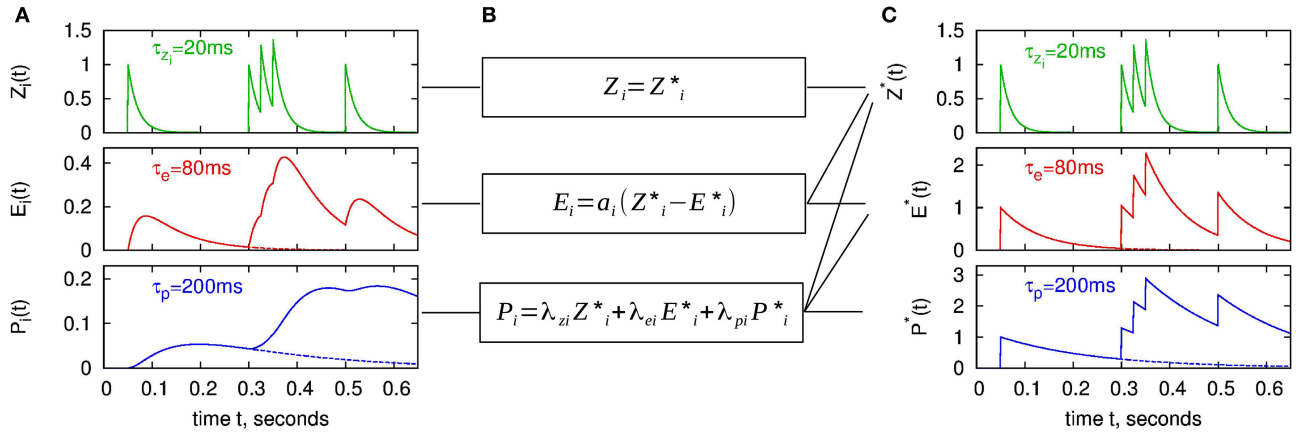

FIGURE 3 | BCPNN dynamics in two different representations for an example spike train. (A) Presynaptic traces according to the original model formulation (Z_i, E_i, P_i), as expressed in Equations (7–9). **(C)** "Exponential" traces Z_i^*, E_i^*, P_i^* used for the analytical II solution with minimal number of calculations, according to Equations (32–34). Dashed lines in **A** and **C** denote spike response kernels, i.e., responses to a single spike. **(B)** Transformation between the two different representations (Equations 29, 30, 46).

$$= \sum_{t_i^f} Z_j(t_i^f) e^{-(t-t_i^f)(\frac{1}{\tau_{z_i}} + \frac{1}{\tau_{z_j}})} \Theta(t - t_i^f)$$

$$+ \sum_{t_j^f} Z_i(t_j^f) e^{-(t-t_j^f)(\frac{1}{\tau_{z_i}} + \frac{1}{\tau_{z_j}})} \Theta(t - t_j^f) \quad . \qquad (20)$$

For Equation (20) we employed the fact that for the spike response of a presynaptic spike at time t_i^f, we can neglect the contribution of future postsynaptic spikes with $t_j^f > t_i^f$, and vice versa. Hence, similar to the Z_i and Z_j, the product Z_iZ_j can be written by means of the spike kernels ζ_{ij}:

$$Z_iZ_j = \sum_{t_i^f} Z_j(t_i^f)\zeta_{ij}(t - t_i^f) + \sum_{t_j^f} Z_i(t_j^f)\zeta_{ij}(t - t_j^f),$$

$$\zeta_{ij}(t) = e^{-\frac{t}{\tau_{z_{ij}}}} \Theta(t). \qquad (21)$$

In contrast to Z_i and Z_j, where all spikes have equal strength, for Z_iZ_j the spike response of each presynaptic spike t_i^f is scaled by the current value of the postsynaptic Z_j trace, respectively by $Z_i(t_j^f)$ for each postsynaptic spike t_j^f.

As the E_{ij} trace is just a low-pass filtered version of the product Z_iZ_j, we can analogously write the E_{ij} and P_{ij} traces as spike response models:

$$E_{ij}(t) = \sum_{t_i^f} Z_j(t_i^f)\alpha_{ij}(t - t_i^f) + \sum_{t_j^f} Z_i(t_j^f)\alpha_{ij}(t - t_j^f) \quad (22)$$

$$P_{ij}(t) = \sum_{t_i^f} Z_j(t_i^f)\pi_{ij}(t - t_i^f) + \sum_{t_j^f} Z_i(t_j^f)\pi_{ij}(t - t_j^f), \quad (23)$$

with spike response kernels:

$$\alpha_{ij}(t) = a_{ij}\left(e^{-\frac{t}{\tau_{z_{ij}}}} - e^{-\frac{t}{\tau_e}}\right)\Theta(t) \qquad (24)$$

$$\pi_{ij}(t) = a_{ij}\left[b_{ij}\left(e^{-\frac{t}{\tau_{z_{ij}}}} - e^{-\frac{t}{\tau_p^*}}\right) + c\left(e^{-\frac{t}{\tau_p^*}} - e^{-\frac{t}{\tau_e}}\right)\right]\Theta(t) \quad (25)$$

We remark that Equations (20, 22, 23) are ambiguous for the limit case of simultaneous pre- and postsynaptic spikes ($t_i^f = t_j^f$), as it is unclear whether the sampled Z_i and Z_j correspond to the values directly before or after the spikes. This is resolved in Section 2.3.3.

2.3.3. BCPNN solution with reduced operations: analytical II

For the analytical update of the Z, E and P traces derived in Section 2.3.1, we observe that especially the update of P traces is expensive in terms of number of operations. In the presented BCPNN architecture, each MCU has approximately 10,000 inputs and correspondingly as many outputs. It would therefore be of great benefit to reduce the computational cost of the update of the synaptic traces. We achieve this by transforming the BCPNN variables to a new set of state variables that all decay exponentially over time and are only increased when a spike occurs. This is motivated by the spike response model formulation (Section 2.3.2), where the Z_i, E_i, P_i traces are superpositions of the spike response kernels ζ_i, α_i, and π_i, which in turn are linear combinations of the exponential functions $e^{-\frac{t}{\tau_{z_i}}}$, $e^{-\frac{t}{\tau_e}}$, and $e^{-\frac{t}{\tau_p^*}}$. Due to the linearity of the system we can choose these exponentials as new state variables to equally describe the BCPNN dynamics.

This second analytic solution of the BCPNN model is henceforth called *analytical II* in this paper.

2.3.3.1. Presynaptic traces.

For the presynaptic side, we introduce the new state variables Z_i^*, E_i^*, and P_i^*:

$$Z_i^*(t) = \sum_{t_i^f} \mathrm{e}^{-\frac{t-t_i^f}{\tau_{z_i}}} \Theta(t - t_i^f) \tag{26}$$

$$E_i^*(t) = \sum_{t_i^f} \mathrm{e}^{-\frac{t-t_i^f}{\tau_e}} \Theta(t - t_i^f) \tag{27}$$

$$P_i^*(t) = \sum_{t_i^f} \mathrm{e}^{-\frac{t-t_i^f}{\tau_p^*}} \Theta(t - t_i^f) \quad , \tag{28}$$

which can be used to express Z_i, E_i, and P_i:

$$Z_i(t) = Z_i^*(t) \tag{29}$$

$$E_i(t) = a_i \left(Z_i^*(t) - E_i^*(t)\right) \tag{30}$$

$$P_i(t) = a_i \left[b_i \left(Z_i^*(t) - P_i^*(t)\right) + c \left(P_i^*(t) - E_i^*(t)\right)\right] \quad . \tag{31}$$

The time course of the new state variables as a response to an example spike train is shown in **Figure 3C**. Note that we have introduced Z_i^* although it is identical to Z_i in order to emphasize the concept of the new representation with exponentially decaying state variables.

Instead of performing an event-based update of the original state variables Z_i, E_i, and P_i, we can update Z_i^*, E_i^*, and P_i^*: Given that there is no spike between t^{last} and t, the state evolves from t^{last} to t, with $\Delta t = t - t^{\text{last}}$, as:

$$Z_i^*(t) = Z_i^*(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_{z_i}}} + S_i(t) \tag{32}$$

$$E_i^*(t) = E_i^*(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_e}} + S_i(t) \tag{33}$$

$$P_i^*(t) = P_i^*(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} + S_i(t) \tag{34}$$

Thus, between any two times we only have to calculate the exponential decay with τ_{z_i}, τ_e, and τ_p^*. At a new spike, we add 1 to all of the new state variables, compared to the classical lazy model, where only Z_i is increased (cf. **Figure 3**). Of course, equivalent new state variables and the same updating scheme can be used for the postsynapic side.

2.3.3.2. Synaptic traces. For updating the synaptic variables, an analogy can be made to the presynaptic traces. Again, we introduce new state variables E_{ij}^* and P_{ij}^*:

$$E_{ij}^*(t) = \sum_{t_i^f} Z_j(t_i^f) \mathrm{e}^{-\frac{t-t_i^f}{\tau_e}} \Theta(t - t_i^f) + \sum_{t_j^f} Z_i(t_j^f) \mathrm{e}^{-\frac{t-t_j^f}{\tau_e}} \Theta(t - t_j^f) \tag{35}$$

$$P_{ij}^*(t) = \sum_{t_i^f} Z_j(t_i^f) \mathrm{e}^{-\frac{t-t_i^f}{\tau_p^*}} \Theta(t - t_i^f) + \sum_{t_j^f} Z_i(t_j^f) \mathrm{e}^{-\frac{t-t_j^f}{\tau_p^*}} \Theta(t - t_j^f) \tag{36}$$

These, together with Z_i^* and Z_j^*, can be used to express E_{ij} and P_{ij}:

$$E_{ij}(t) = a_{ij} \left(Z_i^*(t) Z_j^*(t) - E_{ij}^*(t)\right) \tag{37}$$

$$P_{ij}(t) = a_{ij} \left[b_{ij} \left(Z_i^*(t) Z_j^*(t) - P_{ij}^*(t)\right) + c \left(P_{ij}^*(t) - E_{ij}^*(t)\right)\right] \tag{38}$$

We first consider the event-based update of the new synaptic state variables E_{ij}^* and P_{ij}^* for a presynaptic spike only (which is equivalent to a postsynaptic spike only). The case of simultaneous pre- and postsynaptic spikes is treated separately afterwards. In order to advance E_{ij}^* and P_{ij}^* from their last updated time t^{last} to t, with $\Delta t = t - t^{\text{last}}$ and no spike within this interval, the update goes as follow:

$$E_{ij}^*(t) = E_{ij}^*(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_e}} + S_i(t) \cdot Z_j(t) \tag{39}$$

$$P_{ij}^*(t) = P_{ij}^*(t^{\text{last}}) \cdot \mathrm{e}^{-\frac{\Delta t}{\tau_p^*}} + S_i(t) \cdot Z_j(t), \tag{40}$$

i.e., E_{ij}^* and P_{ij}^* decay exponentially from their last states and, for the case of a presynaptic spike t_i^f at time t, increase by the sampled postsynaptic $Z_j(t)$ trace. Here lies the difference to the presynaptic update, where each spike has the same effect, whereas the synaptic E_{ij}^* and P_{ij}^* traces are increased depending on the current Z_j value of the postsynaptic side, as the synaptic traces keep track of the *overlap* of pre- and postsynaptic activity.

The case of concurrent pre- and postsynaptic spikes is not well defined in the formulas for E_{ij}^* and P_{ij}^* (Equations 35, 36) and in the spike response model formulation (Equations 22, 23). Therefore, we turn back to the product $Z_i Z_j$, which at simultaneous pre- and postsynaptic spikes is increased by

$$\Delta_{ij} = Z_i^+ Z_j^+ - Z_i^- Z_j^- \quad . \tag{41}$$

Here Z_i^- (Z_j^-) denotes the Z-trace before the evaluation of a presynaptic (postsynaptic) spike, and Z_i^+ (Z_j^+) after the evaluation:

$$Z_i^+ = Z_i^- + S_i \quad , \quad Z_j^+ = Z_j^- + S_j \quad , \tag{42}$$

where S_i (S_j) is only non-zero if there is a presynaptic (postsynaptic) spike at the current time. Inserting Equation (42) into Equation (41) yields

$$\Delta_{ij} = (Z_i^- + S_i)(Z_j^- + S_j) - Z_i^- Z_j^- \tag{43}$$

$$= S_i Z_j^- + S_j Z_i^- + S_i S_j \tag{44}$$

$$= S_i Z_j^- + S_j Z_i^+ \quad . \tag{45}$$

The increment Δ_{ij} not only describes the change of $Z_i Z_j$, but also applies to updates for the new synaptic traces E_{ij}^* and P_{ij}^*. Equation (44) can be used when both spikes are evaluated *synchronously*, Equation (45) when both spikes are evaluated *consecutively*, i.e., when first the presynaptic spike is processed (first summand), and afterwards the postsynaptic spike (second summand). For the event-based benchmark simulations (Sections 2.2.2 and 3.1.2),

where all spikes are discretized to multiples of dt, the latter strategy for Δ_{ij} is used for the update in the synapse array: first all presynaptic spikes are evaluated, then all postsynaptic spikes.

2.3.3.3. Initialization of exponential state variables. This section explains how to set the initial values of the new state variables (Z^*, E^*, P^*) from a given set of Z, E, P traces. Therefore, we first shorten the transformation formula of P_i (Equation 31) with new coefficients as:

$$P_i = \lambda_{zi} Z_i^* + \lambda_{ei} E_i^* + \lambda_{pi} P_i^* \tag{46}$$

$$\lambda_{zi} = a_i b_i \quad , \quad \lambda_{ei} = -a_i c \quad , \quad \lambda_{pi} = a_i (c - b_i) \quad . \tag{47}$$

For brevity, we have left out the time dependence of the states. The equivalent simplification can be applied for the postsynaptic traces. Similarly, the synaptic traces (Equations 37, 38) can be written as

$$E_{ij} = a_{ij} \left(Z_i^* Z_j^* - E_{ij}^* \right) \tag{48}$$

$$P_{ij} = \lambda_{zij} Z_i^* Z_j^* + \lambda_{eij} E_{ij}^* + \lambda_{pij} P_{ij}^* \quad , \tag{49}$$

with coefficients

$$\lambda_{zij} = a_{ij} b_{ij} \quad , \quad \lambda_{eij} = -a_{ij} c \quad , \quad \lambda_{pij} = a_{ij} \left(c - b_{ij} \right) \quad . \tag{50}$$

To turn the set of Z, E, P variables into the new state variables (Z^*, E^*, P^*), the following reverse transformation holds:

$$Z_i^* = Z_i \tag{51}$$

$$E_i^* = -\frac{E_i}{a_i} + Z_i^* \tag{52}$$

$$P_i^* = \frac{1}{\lambda_{pi}} \left(P_i - \lambda_{zi} Z_i^* - \lambda_{ei} E_i^* \right) \tag{53}$$

Note that the transformation has to be performed in the above order. The synaptic values are set as follows:

$$E_{ij}^* = -\frac{E_{ij}}{a_{ij}} + Z_i^* Z_j^* \tag{54}$$

$$P_{ij}^* = \frac{1}{\lambda_{pij}} \left(P_{ij} - \lambda_{zij} Z_i^* Z_j^* - \lambda_{eij} E_{ij}^* \right) \tag{55}$$

2.4. BENCHMARKS

To validate our implementation of the BCPNN we used several benchmarks, targeting either simulation run time or accuracy. As infrastructure for the simulations we used a cluster with Intel®Xeon®CPU E5-2690 2.90 GHZ. All benchmarks were implemented in C++ and compiled with GCC 4.7.1. All simulations were single-threaded. The time constants and other BCPNN parameters used for the benchmarks are listed in **Table 1**.

Table 1 | Parameters used in the execution time and accuracy benchmarks.

Synapse model	
Parameters	$\tau_{z_i} = 10$ ms presynaptic Z trace time constant
	$\tau_{z_j} = 15$ ms postsynaptic Z trace time constant
	$\tau_e = 20$ ms E trace time constant
	$\tau_p = 1000$ ms P trace time constant
	$\kappa = 1$ learning rate
	$\epsilon = 0.001$ minimum activity

Note that the values represent only one possible parameter set. Plausible ranges for the time constants are given in the text (Section 2.1.1). The execution time is not affected by the choice of the parameters, but, of course, the accuracy results may change when using different parameters.

2.4.1. Simulation run time

To compare the computational cost of the different update strategies, we simulated the synaptic dynamics of a full hypercolumn with 10,000 inputs and 100 MCUs, see **Figure 2**. For both pre- and postsynaptic units we use independent Poisson spike trains, which are pre-generated and then read from a file to the main program, so that equal spike trains are used for the different update strategies. The simulation runs for 10 s, the Poisson rate is swept over a range of 0.01–100 Hz. For each rate and update strategy we assess the execution time per simulated second as the average of 5 runs with different random seeds. Although independent Poisson spike trains for the pre- and postsynaptic units will not be the case in realistic BCPNN applications including learning and retrieval of patterns, they sufficiently model the probabilistic nature of the MCUs and are thus favorable compared to regular spike trains. In order to measure only the computational cost of the synaptic updates, the stochastic MCUs are not simulated in this benchmark. However, for a fair comparison of the update strategies, we calculate the support value s_j (Equation 3) for all postsynaptic units at each time step, so that all β_j are calculated at every time step, and the weights w_{ij} are computed whenever a spike of presynaptic unit i arrives.

2.4.2. Accuracy comparison

As many published results are based on an explicit Euler method (see e.g., Johansson and Lansner, 2007; Berthet et al., 2012; Kaplan and Lansner, 2014), we compare this numerical method to an exact analytical one in Section 3.2.2. Furthermore, we investigate the influence of using fixed-point operands with different number of bits instead of floating point numbers with double precision. For this purpose, we implemented a single BCPNN synapse in C++ with templates allowing the comparison of different number formats, making use of an in-house developed fixed-point library.

As stimuli for the BCPNN synapse we generated pre- and postsynaptic spike trains according to a homogeneous Poisson process with rate r. For the accuracy benchmarks not only the update frequency is important but also that different dynamical ranges of the BCPNN variables can be triggered, which requires different levels of correlation. To achieve that, we follow Kuhn et al. (2003) and create pre- and postsynaptic Poisson spike trains that share a fraction of c correlated spike times. Therefore, we create one correlated Poisson spike train with rate $c \cdot r$, and two independent Poisson spike trains with rate $(1 - c) \cdot r$ for the pre- and postsynaptic side. The correlated spike times are then added

to both independent spike trains. To avoid a systematic zero-lag between pre- and postsynaptic spike times, the correlated spike times of the postsynaptic side are jittered according to a Gaussian distribution with standard deviation $\sigma = 5$ ms.

We run multiple simulations to investigate the effects of the Euler method and fixed-point operands, respectively. For each accuracy setting, stimuli are generated using 11 correlation factors c ranging from 0 to 1 in intervals of 0.1. For each of the different correlation factors, 10 different seeds are used for the Poisson processes, resulting in 110 simulations per accuracy setting. The stimuli are generated with an average rate of 1 Hz and the duration of each simulation is 1000 s. For the Euler method, spike times are set to multiples of the time step to avoid time discretization errors (Henker et al., 2012). For fixed-point operands, spike times are generated with a resolution of 0.01 ms.

To assess the accuracy of the different implementations, we consider absolute errors e_{abs}:

$$e_{\text{abs}} = |x - \hat{x}| \quad , \tag{56}$$

where x denotes the exact value (analytical solution with floating point double precision) and $\hat{x}$ is the approximation (either the Euler solution or the analytical solution with fixed-point operands). By point wise comparing each of the state variables (e.g., w_{ij}, P_{ij} ...), the accuracy can be assessed. The mean absolute error $\overline{e_{\text{abs}}}$ is the average of the single absolute errors determined at each time of a pre- or postsynaptic spike over all simulation runs. The normalized mean absolute error (NMAE) is the mean absolute error divided by the range of observed values x:

$$\text{NMAE} = \frac{\overline{e_{\text{abs}}}}{x_{\max} - x_{\min}} \quad , \tag{57}$$

which allows to compare the accuracy of several variables with different scales.

3. RESULTS

The two analytic solutions for spike-based BCPNN derived in Section 2.3 allow an efficient event-driven simulation of BCPNNs. In Section 3.1 we investigate how this reduces the computational footprint of BCPNN learning both formally and empirically. Aiming also for a small memory footprint, we evaluate the use of fixed-point numbers for the storage of BCPNN state variables, and compare the introduced discretization errors with the errors caused by the fixed step size simulation with the Euler method (Section 3.2).

3.1. COMPARISON OF SIMULATION STRATEGIES

In this section we compare the computational efficiency of the two analytical solutions of the BCPNN equations against each other and to the commonly used fixed step size implementation with the Euler method. We also investigate the benefit of using look-up tables for exponential decays in the analytical II method.

3.1.1. Number of operations

We start with a formal comparison between the two analytical update solutions by counting the steps of calculation required for an event-based update in each representation. Therefore, we categorize the operation into three classes: ADD combines both additions and subtractions, MUL stands for multiplications and divisions, EXP for calculations of the exponential function, and LOG for the natural logarithm.

Table 2 lists the number of operations needed by the analytical I and analytical II methods for different tasks: For the update of the presynaptic state variables (Z_i, E_i, P_i resp. Z_i^*, E_i^*, P_i^*) at an incoming spike, most notably, the analytical II method requires 6 MUL and 3 ADD operations less than the analytical I method. Instead, when the P_i value is retrieved, e.g., to calculate the synaptic weight w_{ij}, the analytical I method requires zero operations, while the analytical II method requires 2 ADD and 3 MUL operations to calculate P_i from Z_i^*, E_i^*, and P_i^*. Here, the difference between the two strategies manifests: while the analytical II is more efficient when the states are updated, it requires additional operations to determine the original states. Nevertheless, when adding up the counts of both tasks (pre-update and retrieval of P_i), e.g., when β_j is updated after a postsynaptic spike, the analytical II is still much more efficient than the analytical I method.

Table 2 | Arithmetic operations per task for different analytical update methods.

Task	Operation	Analytical I	Analytical II
Pre-update		Equations (7–9)	Equations (32–34)
	ADD	7	3
	MUL	12	6
	EXP	3	3
Retrieve P_i			Equation (46)
	ADD	–	2
	MUL	–	3
Syn-update		Equations (12, 13)	Equations (39, 40)
	ADD	6	2
	MUL	13	5
	EXP	3	2
Retrieve P_{ij}			Equation (49)
	ADD	–	2
	MUL	–	4
Update of w_{ij} at pre-spike	ADD	22	14
	MUL	39	29
	EXP	9	8
	LOG	1	1
Update of β_j at post-spike	ADD	8	6
	MUL	12	9
	EXP	3	3
	LOG	1	1

ADD, additions and subtractions; MUL, multiplications and divisions; EXP, computations of exponential function; LOG, natural logarithm. The operation counts of the analytical I method correspond to optimized versions of the referenced formulas using pre-calculated coefficients and intermediate steps. The different tasks are further specified in the text.

Similar results are found for the update of the synaptic state variables (E_{ij}, P_{ij} resp. E^*_{ij}, P^*_{ij}), where the advantage of the analytical II over the analytical I is even larger, cf. **Table 2**. Again, the analytical II strategy needs additional steps of computation for the retrieval of P_{ij}. For the typical case of a presynaptic spike, where all traces and the weight are updated (task "update of w_{ij} after pre-spike") and which includes the retrieval of all P-traces, the analytical II requires considerably less operations than the analytical I method. Note that the speedup of the analytical II is even higher when processing a post-synaptic spike, as then the weight needs not be calculated and thus the P traces need not be retrieved.

If we consider an array of BCPNN synapses, as in a hypercolumn unit of the reduced cortex model (**Figure 2**), where the pre- and postsynaptic traces are handled at margins of the array, it is the update and retrieval of the *synaptic* BCPNN state variables that make up the majority of the calculations. Assuming equal mean firing rates for the pre- and postsynaptic units, the P_{ij} values need to be retrieved on average only at every second spike event. In that case, the analytical II method requires roughly half the number of basic arithmetic operations (ADD and MUL) of the analytical I method, but only slightly less calculations of the natural exponential function.

3.1.2. Simulation run time

As a complement to the formal comparison, we measured the simulation run time required to simulate the update of synapses of one HCU with 10,000 inputs and 100 outputs for the different update strategies. The results for a sweep over the Poisson firing rates of the inputs and outputs, which is described in detail in Section 2.4.1, are shown in **Figure 4A**. As expected, for the fixed step size simulation with explicit Euler method and $dt = 1$ ms the execution time depends only slightly on the spike frequency: It takes ≈ 2.4 s to simulate 1 s of the network for firing rates up to 10 Hz, only for higher rates the run time increases significantly, which can be attributed to the more frequent calculation of synaptic weights. In contrast, for the event-based methods the execution time strongly depends on the firing activity: For very low spike rates, there is a baseline computational cost that can be attributed to the calculation of all postsynaptic biases β_j and support values s_j (Equation 3) at every time step (cf. Section 2.4.1). For Poisson rates of 0.1 Hz and higher, the execution time scales linearly with the firing rate. The update strategy with reduced operations (analytical II, green curve) clearly outperforms the conventional analytical update (analytical I, blue curve). For a typical average firing rate of 1 Hz of MCUs in a HCU (cf. Lansner et al., 2014) the analytical II strategy is more than 3 times faster than the real-time dynamics of the model, while the analytical I update runs approximately at real time. We remark that we optimized the C++ code of the analytical II update as good as possible, while the analytical I code is not optimized to the end. Thus, the results of latter can not be taken as final and should rather be interpreted as an intermediary result.

We compare the run time of the event-based methods directly to the fixed step size simulation in **Figure 4B**. For low spiking activity, the event-based methods are up to 100 times faster than the fixed step size method. At 1 Hz the analytical II strategy (green curve) runs more than 8 times faster than the simulation with Euler. Only for firing rates higher than 20 Hz the fixed step size approach is competitive with, respectively faster than the analytical II method.

Additional results for a 0.1 ms time discretization are provided in Appendix A4, showing a much higher speedup of event-driven methods against the fixed step size method.

3.1.3. Look-up tables for exponential functions

In another simulation we investigated the benefit of using look-up tables (LUTs) for the exponential functions instead of computing the exponential at each event. This is motivated by the number of exponential decays calculated per update (cf. **Table 2**), as well as by a profiling of the implemented C++ program which shows that a huge amount of simulation time is spent in the computation of the exponential function. Look-up tables are especially

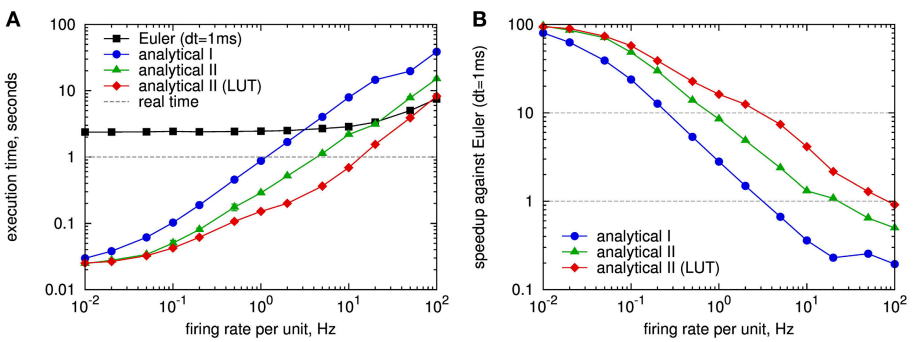

FIGURE 4 | Speed comparison of different simulation strategies for spike-based BCPNN: fixed step size simulation with explicit Euler method with 1 ms time step (Euler, black curve), event-driven simulation with analytical update (analytical I, cf. Section 2.3.1, blue) and analytical update with exponential state variables (analytical II, cf. Section 2.3.3), with and without using look-up tables (LUTs) for the exponential function (red, resp. green). **(A)** Execution time for simulating a full hypercolumn unit with 1 million BCPNN synapses for 1 s with different Poisson firing rates applied to both pre- and postsynaptic units of the HCU (cf. **Figure 2**). **(B)** Speedup of event-based simulation methods with respect to the fixed step size simulation with Euler method in **A**. Look-up tables were implemented for the exponential decay of type $\exp(-\frac{\Delta t}{\tau})$ for the time constants $\tau_{z_i}, \tau_{z_j}, \tau_e, \tau^*_p$. Each LUT had 3000 entries in steps of 1 ms.

beneficial in the used event-driven simulation (Section 2.2.2) where spike times are restricted to multiples of the time step dt. Calculations of the form

$$LUT(N, \tau, dt) = \mathrm{e}^{-\frac{N \cdot dt}{\tau}} \tag{58}$$

are performed very often, where N is the number of time steps that have elapsed since the last update, and τ is one of the four involved time constants $\tau_{z_i}, \tau_{z_j}, \tau_e, \tau_p^*$. In a modified version of the analytical II implementation, we create look-up tables of Equation (58) for the four time constants, each with L entries for $N = 1 \ldots L$. Only if the number of elapsed time steps between two updates is larger than L, the exponential function Equation (58) is computed on demand.

The results for using look-up tables in the analytical II method are included in **Figure 4**: The implementation with look-up tables (red curve) speeds up the simulation for Poisson rates starting from 0.1 Hz, and is up to 3 times faster than the version without LUTs at 10 Hz spiking activity. Here, the size of the LUTs was chosen as $L = 3000$, covering update intervals up to 3 s, so that for a Poisson rate of 1 Hz on average 95 % of the inter spike intervals are handled by the look-up table. For the typical case of 1 Hz the LUT implementation is 1.9 times faster than the one without LUTs, 6.6 times faster than real time, and 16 times faster than the fixed step size simulation with explicit Euler method. For a wide spectrum of tested firing rates the analytical II solution with look-up tables is much more efficient than the fixed step size simulation with Euler, only for a firing rate of 100 Hz the latter performs slightly better (**Figure 4B**), so that in practical situations the fixed step size method becomes dispensable for the simulation of the abstract BCPNN cortex model.

3.2. FIXED-POINT NUMBERS FOR BCPNN TRACES AND THEIR ACCURACY

To store all traces of the 1 million BCPNN synapses of a full HCU, one requires more than 12 MB assuming a single precision floating point number occupying 4 byte for each state variable (Lansner et al., 2014). Targeting an implementation of the BCPNN model on neuromorphic hardware, the use of fixed-point numbers can reduce the number of computational and storage resources, possibly at the price of loosing precision. Therefore, we investigate the accuracy of using fixed-point operands to store the state variables in the event-based simulation with analytical II method, and compare it to the accuracy of the fixed step size simulation with the Euler method.

3.2.1. Value range estimation

For a fixed-point implementation of the BCPNN model, it is important to determine an upper bound of each state variable. This bound can be used to normalize the variables, so that an identical fixed-point representation can be used for all.

For a single exponential trace, be it the Z_i and Z_j traces in the standard analytical solution or the state variables of the analytical II solution, an upper bound can be calculated using a regular spike train with maximum rate $r_{\max}$. The value of this rate may be derived from the units' refractoriness period or as a multiple of the mean unit firing rate, accounting for short-term firing rate

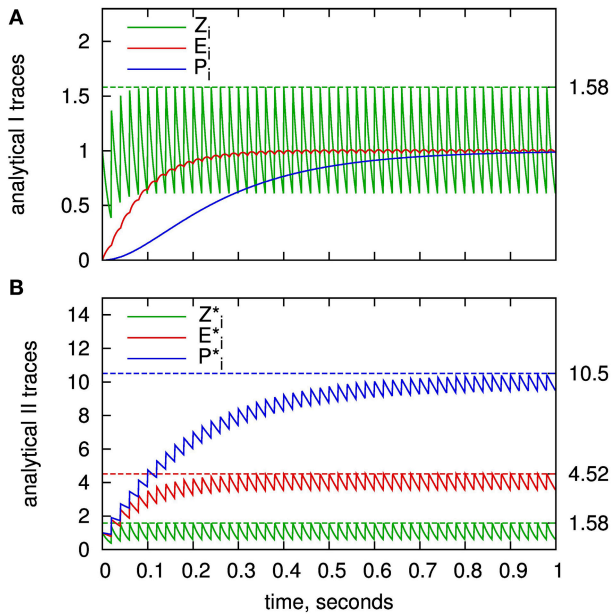

FIGURE 5 | Value range estimation of the BCPNN states variables: Traces of the Z, E, and P states for a regular spike train with frequency of 50 HZ. **(A)** Presynaptic traces according to the standard analytical solution (analytical I), as expressed in Equations (7–9). **(B)** Exponential traces of the improved analytical solution analytical II, Equations (32–34). Analytically calculated limits according to Equation (59) are plotted as dashed lines. Time constants used here: $\tau_{z_i} = 20\,\mathrm{ms}, \tau_e = 80\,\mathrm{ms}$ and $\tau_p^* = 200\,\mathrm{ms}$.

fluctuations. The upper bound can be calculated from the equilibrium state, where exponential decay and instantaneous increase at a spike equalize:

$$\begin{aligned} Z_i(t_n) &\stackrel{!}{=} Z_i(t_{n+1}) = Z_i(t_n) \cdot \mathrm{e}^{-1/(r_{\max} \cdot \tau_{z_i})} + S_i \\ \Rightarrow Z_{i,\max} &= \frac{S_i}{1 - \mathrm{e}^{-1/(r_{\max} \cdot \tau_{z_i})}} \end{aligned} \tag{59}$$

The upper bounds of the presynaptic traces are illustrated in **Figure 5A**. They very closely match the actual maximums of the traces according to the employed regular spike train. For the traces of E_i and P_i, the same maximum as for Z_i can be used in good approximation. This is motivated from the differential equations of the model given in **Figure 1**: The worst-case assumption for Z_i from the maximum calculation would be a constant value of $Z_{i,\max}$. Given this input, the trace of E_i would approach the same value. The same argument in turn holds for P_i.

For the traces of the analytical II solution, Z_i^*, E_i^*, and P_i^*, Equation (59) can be used with according time constants. For $r_{\max} \cdot \tau \gg 1$, the maximum can be approximated as $r_{\max} \cdot \tau$ for an increment of $S_i = 1$. The highest absolute value is reached for the longest time constant, which is $\tau_p = 1000\,\mathrm{ms}$ in our example parameter set. Assuming a refractoriness period of 1 ms, the worst-case upper bound would be $P_{i,\max}^* \approx 1000$. For a fixed-point implementation, a width of 10 integer bits would be sufficient to avoid any unwanted saturation or overflows. It can be

expected that the actual maximum of P_i^* is significantly lower as it is extremely unlikely that a neuron (resp. a MCU) fires every 1 ms for a multitude of spikes. Thus, for a specific benchmark, a lower bound may be determined from simulation.

3.2.2. Accuracy comparison

We ran multiple simulations to investigate the effects of the Euler method and fixed-point operands, respectively. For each accuracy setting, a single BCPNN synapse was stimulated by pre- and postsynaptic Poisson spike trains of 1 Hz average rate. We applied different levels of correlation between pre- and postsynaptic spike trains in order to generate wide value ranges of the BCPNN variables, especially for w_{ij}. The simulation setup is described in detail in Section 2.4.2.

The accuracy results for both Euler method and fixed-point operands are shown in **Figure 6**. As accuracy measure, we assess the normalized mean absolute error (NMAE) as described in Section 2.4.2. To get an impression of the variable ranges in the simulations, we give their average, minimum and maximum in **Table 3**. Note that we only show the errors for w_{ij} and β_j (and not for the Z,E,P traces), as these are the only BCPNN variables that affect the activation of the postsynaptic units. As expected, the Euler method exhibits a linearly increasing accuracy with decreasing step size (**Figure 6A**). The accuracy is worse for the synaptic weight w_{ij} than for the bias β_j, as the w_{ij} error is affected by the errors of P_i,P_j, and P_{ij}, while β_j only depends on the accuracy of P_j. For 1 ms step size, which we used for the execution time benchmarks, the normalized mean absolute error of the synaptic weight lies far below 1 %. A reason for this relatively small error might be the exponentially decaying dynamics of the BCPNN variables, which keeps the accumulation of errors low.

For fixed-point operands, we used calculation with floating point precision, but quantized each intermediate result for a state variable to a fixed number of fractional bits. For the time constants and coefficients (Equations 47, 50) we used the highest available fixed-point precision (32 fractional bits) to minimize computational errors. This emulates the case that state variables are stored with limited precision to reduce storage space, but the arithmetic operations are designed such that they do not introduce additional numerical errors. Quantization errors can be modeled as a noise source with amplitude 2^{-b}, where b is the number of fractional bits. All errors scale according to this noise source (compare dashed line in **Figure 6B**). Again, the accuracy is higher for β_j than for w_{ij}, but now the ratio between w_{ij} and β_j errors is larger than in the Euler simulation.

Comparing these results answers the question what fixed-point operand resolution is required in our optimized analytical solution to achieve at least the same accuracy as state-of-the-art Euler methods. This can be derived from curves with equal mean absolute error, as shown in the lower diagram of **Figure 6C**. In terms of scaling, Euler method and fixed-point operands compare as

$$\overline{e_{\text{abs}}} = A_{\text{Euler}} \cdot dt = A_{\text{fixed}} \cdot 2^{-b} \,, \tag{60}$$

where dt is the step size of the Euler method and A_{Euler}, A_{fixed} are variable-specific constants. The corresponding line $dt = 2^{-b}$ is drawn as dashed line in the diagram. As expected from the previous results, the single errors follow this line, shifted by an offset. For a time step of $dt = 0.1$ ms 16 fractional bits or less are

Table 3 | Measured ranges of BCPNN state variables in accuracy simulations.

Variable	Mean	Min	Max
P_i	0.010	0.001	0.066
P_j	0.015	0.001	0.097
P_{ij}	0.0028	0.000	0.898
w_{ij}	1.57	−6.75	5.35
β_j	−4.38	−6.21	−2.32

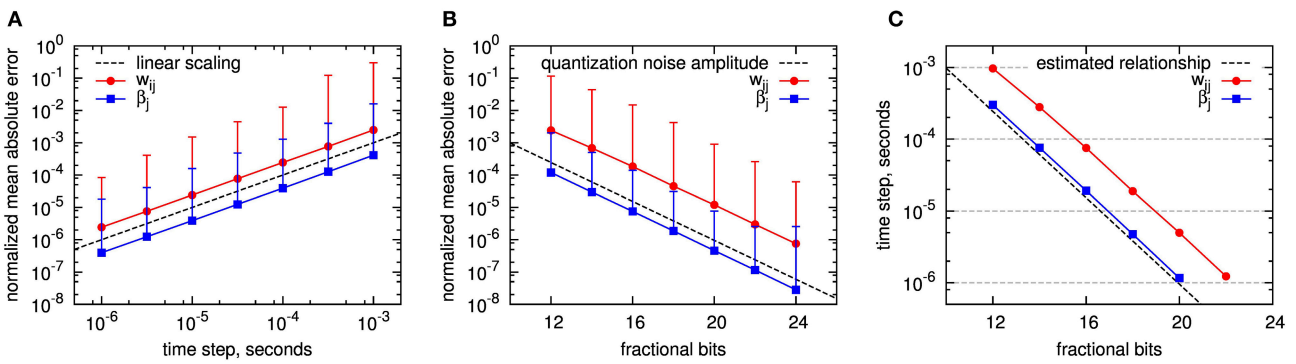

FIGURE 6 | Accuracy of fixed step size simulation with Euler method and event-driven analytic simulation using fixed point operands. The accuracy of w_{ij} and β_j is assessed by the normalized mean absolute error taken over a large set of experiments with the exact analytical solution as reference, see text for details. **(A)** Simulation with Euler, dependent on step size. The dashed line shows the linear scaling: $y(dt) = dt \cdot \text{s}^{-1}$. **(B)** Analytical solution with event-driven update (analytical II) using fixed-point representation with different counts of fractional bits. The dashed line shows the quantization noise amplitude: $y(b) = 2^{-b}$. The error bars in **A** and **B** denote the normalized maximum absolute error recorded within all simulations per setup. **(C)** Comparison between the errors introduced by the Euler method and the use of fixed-point numbers with limited number of fractional bits: For w_{ij} and β_j the location of *equal* mean absolute errors is plotted, depending on the step size for the Euler method, respectively the number of fractional bits of the fixed-point implementation. Dashed line: estimated relationship according to Equation (60).

required to achieve at least the same accuracy in all variables. A number of integer bits is required in addition to represent values greater than one. As shown in Section 3.2.1, a maximum of 10 integer bits is required in a worst-case scenario for the employed parameter set.

For the simulation of the reduced modular model of the cortex described in Section 2.1.2, for which a 1 ms time step seems to provide sufficient results (Lansner et al., 2014), only 12 fractional bits, and thus at maximum 22 bits in total, are needed to ensure equal or better accuracy compared to using the Euler method. Hereby, the required memory per state variable decreases by almost one third compared to using single precision floating point numbers.

Considering a 0.1 ms time step and a 64 bit floating point representation, which is commonly used in state-of-the art neural network simulations, fixed-point numbers with less than 32 bits yield competitive accuracy, so that the memory footprint reduces even by more than a half.

4. DISCUSSION

In this paper we derived two analytic solutions for the spike-based BCPNN learning rule. They enable an efficient event-driven simulation of spiking neural networks employing this learning rule, such as the reduced modular model of cortex (Lansner et al., 2014). The advantages of using an analytic over a fixed step size numeric solution are twofold: Firstly, it enables an event-driven update of the variables, and thereby significantly speeds up synaptic plasticity when interspike intervals are long compared to simulation time resolution. Secondly, it increases the precision of the calculations compared to fixed step size methods. Both aspects can be utilized for allocating resources in an existing hardware system efficiently or in conceiving a neuromorphic system based on the BCPNN computational paradigm.

4.1. CLASSIFICATION AND LIMITATIONS OF OPTIMIZATION

In our simulations including 1 million BCPNN synapses with pre- and postsynaptic activity at 1 Hz, we were able to reduce the execution time by a factor of 16 compared to the conventional fixed step size simulation with explicit Euler. One hypercolumn unit of the reduced cortex model was simulated more than 6 times faster than real time on a single CPU. Several factors are responsible for that speedup:

By employing the analytical I solution of the BCPNN model, the *event-driven simulation* becomes feasible and clearly defeats the time-driven simulation at the chosen working point of 1 Hz firing rate. In general, the event-driven approach is mostly advantageous over the time-driven approach when the firing rates are low and connectivity is sparse (Brette et al., 2007). Hence, as long as the inter-spike intervals are large compared to the simulation step size, the analytic event-driven simulation can effectively reduce the execution time of spiking neural networks, independent whether the BCPNN synapses connect single neurons or more abstract units like in the cortex model.

The *analytical II* solution requires on average only half of the basic arithmetic operations of the conventional analytical I solution for an event-based update, and slightly less calculations of the exponential function. Here, the computational cost is reduced by representing the same BCPNN dynamics with a set of exponentially decaying state variables, which is possible due to the linearity of the system. A similar approach has been taken by Brette (2006) for the exact simulation of leaky integrate-and-fire neurons with synaptic conductances, albeit with the restriction of equal excitatory and inhibitory synaptic time constants. Quite the opposite, the only limitation for the BCPNN synapse model is that the decay time constants of the three low pass filtering stages must differ. Nevertheless, when a specific network model requires equal time constants, one can still switch to the analytical I solution provided in Appendix A2, or try slightly different parameters.

The usage of *look-up tables* for the frequent calculation of exponential decays can further accelerate the simulation by a factor of 2 or 3. Precalculated look-up tables are a common tool in event-driven neural network simulations to reduce the cost for the calculation of complex functions (Brette, 2006; Ros et al., 2006). For BCPNN, LUTs for the exponential decay are beneficial as long as the time constants are homogeneous and do not vary from synapse to synapse. In our hybrid simulation of a hypercolumn unit, where spikes are discretized to multiples of the simulation step size, look-up tables not only accelerate the simulation, but also provide the same accuracy as the solution without LUTs. For simulations with arbitrary update intervals, linear interpolation can be used to achieve almost exact results (Brette, 2006). Alternatively, for the case of the exponential function, the computation can be split into two steps, e.g., by first retrieving the EXP separately for the integer and fractional bits of the exponent, and then multiplying the two obtained results. There remains the question for the optimal size and resolution of the look-up tables, which must be chosen depending on the used hardware platform (available memory, cache size) and the inter spike interval distributions of actual network models.

The optimizations presented in this paper focus on reducing the computational footprint for the spike-based BCPNN learning rule: In our benchmarks we have considered either a single synapse or an array of synapses, but not the dynamics of neurons or the MCUs. The efficient simulation of large recurrent networks with many HCUs entails many new issues, e.g., the distribution of hypercolumns across compute nodes and memory, the communication of spikes between HCU or the buffering of spikes, and gives rise to separate studies that are clearly out of scope of this paper.

4.2. ACCURACY

Fixed-point operands can reduce the memory footprint with the drawback of loosing precision compared to a floating point representation. To find the compromise between the two solutions, we assessed the accuracy of using fixed-point operands for the storage of the BCPNN state variables in an event-based simulation with the analytical II method (Section 3.2). The accuracy was compared to the errors introduced by the fixed step size simulation with explicit Euler method using 64 bit floating point numbers, which is commonly used in neural simulation. We found that fixed-point numbers with 22 bits assure equal or better accuracy for all BCPNN variables than the Euler method with 1 ms time step, resp. 26 bits for 0.1 ms time step.

The question remains about which accuracy is necessary in a practical situation. A previous study (Johansson and Lansner, 2004) on using fixed-point arithmetic for BCPNNs showed that an attractor network with 8 bit weights can offer the same storage capacity as an implementation with 32 bit floating point numbers. To achieve this, probabilistic fractional bits were used and the computation of the moving averages (low-pass filters) was performed in the logarithmic domain. Given these results, we speculate that also spike-based BCPNN can be implemented in fixed-point arithmetic with 16 or less bits without loosing computational capabilities, so that the required memory and memory bandwidth can be halved compared to 32 bit floating point numbers used in Lansner et al. (2014).

4.3. NEUROMORPHIC HARDWARE

Our optimizations can be directly incorporated for designing more efficient neuromorphic hardware systems. There are currently several diverse attempts for building large-scale hardware platforms, aiming for a more efficient simulation of large-scale neural models in terms of speed, power or scalability (Schemmel et al., 2012; Hasler and Marr, 2013; Benjamin et al., 2014; Furber et al., 2014; Merolla et al., 2014). As in our analysis, realizing synaptic plasticity is the most resource-demanding task, so that a focus of neuromorphic designs is in efficiently emulating plasticity mechanisms, most often implementing some variant of spike-timing dependent plasticity (STDP, Bi and Poo, 1998; Morrison et al., 2008) in analog or mixed-signal circuitry, see Azghadi et al. (2014) for a review.

An implementation of the BCPNN learning rule requires a stereotypical set of coupled low-pass filters, see **Figure 1**. Implementation of the rule in analog neuromorphic hardware is technically feasible, as there is large knowledge on building leaky integrators (Indiveri et al., 2011), and even the issue of long decay time constants in nanometer CMOS technologies can be resolved, e.g., with switched capacitor techniques (Noack et al., 2014). In this context, our optimized analytic solution offers an interesting alternative to the direct implementation of the original model equations: When using the analytical II solution, the stereotypical low-pass filters are only charged at incoming spikes, in contrast to the continuous coupling in a direct implementation. This alleviates the need for a continuous, variable-amplitude charging mechanism for the E and P traces. On the other hand, charging only at incoming spikes requires a more elaborate calculation of the output values, as present in the analytical II solution. However, this calculation needs to be performed only at spikes as well, allowing e.g., for an efficient implementation with switched-capacitor circuits.

The design of analog neuromorphic circuits is time-consuming and the circuits are affected by parameter variations due to device mismatch. Digital implementations are much less affected by these problems. They may be less energy and area efficient on the level of single elements and they do not allow for direct ion-channel-to-transistor analogies as employed in traditional neuromorphic designs (Hasler et al., 2007). However, they allow to fully utilize the energy efficiency and performance advantages of neural algorithms and modeling approaches, while offering better controllability and scalability.

Several purely digital neuromorphic systems support synaptic plasticity, implemented either on application-specific integrated circuits (Seo et al., 2011), on field-programmable gate arrays (FPGAs) (Cassidy et al., 2013) or a custom multiprocessor system using a larger number of general purpose ARM cores (SpiNNaker system, Furber et al., 2014). Recently Diehl and Cook (2014) showed how general STDP rules can be efficiently implemented on SpiNNaker, despite the system's restriction that synaptic weights can be modified only at the arrival of a presynaptic spike. By adopting their implementation of trace-based STDP, the event-driven spike-based BCPNN in variant analytical I or analytical II can be seamlessly integrated on the SpiNNaker hardware. As we do, Diehl and Cook (2014) use look-up tables for the exponential function; furthermore, SpiNNaker uses fixed-point arithmetic, so that our insights on the accuracy of fixed-point operands may find immediate application.

The event-driven approach is also amenable to state-of-the-art methods for reducing the energy of computation in digital systems. Recent multi-core hardware platforms support fine grained per-core power management, as for example demonstrated on the Tomahawk multiprocessor system-on-chip (MPSoC) architecture (Arnold et al., 2014; Noethen et al., 2014). By changing both the clock frequency and the core supply voltages of each processing element in a dynamic voltage and frequency scaling scheme (Höppner et al., 2012), the hardware performance can be adapted to the performance requirements to solve a particular part of the BCPNN in real time with reduced energy consumption, e.g., by regarding the number of incoming spikes per HCU per simulation step. In addition, within phases of low activity complete processing elements can be shut off to reduce leakage power consumption. Another candidate architecture for energy-efficient neural computation with BCPNNs is the multi-core Adapteva-Epiphany chip (Gwennup, 2011), which is optimized for power-efficient floating point calculations requiring only one fifth of the energy at equal flop rate as the state-of-the-art (but general-purpose) ARM's Cortex-A9 CPU.

Alternatively, spike-based BCPNN can be implemented on novel systems rather than on existing digital systems: For example, one may build dedicated digital hardware for the simulation of the BCPNN cortex model. Such a system containing compact supercomputer functionality can be prototyped in an FPGA with special units for the learning rule or the stochastic minicolumn units, and has therefore only low risk compared to mixed-signal implementations. Recently, Farahini et al. (2014) provided a concept for a scalable simulation machine of the abstract cortex-sized BCPNN model with an estimated power-dissipation of 6 kW in the technology of 2018, which is three orders of magnitudes smaller than for a full-cortex simulation on a supercomputer in comparable technology with 20 billion neurons and 10,000 times more synapses (see also Lansner et al., 2014). They assume the analytical I method for the event-driven updating of the BCPNN traces, and apply floating point units for arithmetic operations. Our work can further promote their performance: By using the analytical II method with look-up tables the computational cost can be further reduced; by moving to

fixed-point arithmetics the required memory and memory bandwidth decreases, so that a low-power real-time simulation of the cortex becomes possible.

4.4. OUTLOOK

Of course, our optimizations can also be used to boost the simulation of spike-based BCPNN on conventional computing systems. For example, already the supercomputer simulations of the reduced cortex model by Benjaminsson and Lansner (2011) showed weak scaling and achieved the real-time operation when simulating one HCU per processor with the fixed step size Euler method ($dt = 1$ ms) and spike-based BCPNN synapses without E traces (the Z traces are directly passed to the P traces). Such large-scale BCPNN simulations are mostly bounded by computation rather than by inter-process communication (Johansson and Lansner, 2007; Lansner et al., 2014), as the firing activity is low and the connectivity is sparse and patchy. Hence, we conjecture that with our approach a speedup factor of 10 or more might be achieved. At the same time, our results can accelerate the simulations of small or medium-scale neural networks employing the spike-based BCPNN learning rule, with applications ranging from olfaction modeling (Kaplan and Lansner, 2014), reward learning (Berthet et al., 2012) to probabilistic inference (Tully et al., 2014). Regardless of whether the BCPNN is implemented in neuromorphic hardware, on a single PC or on supercomputers, the presented optimization through event-driven simulation with look-up tables can boost the success of the BCPNN paradigm as a generic plasticity algorithm in neural computation.

Furthermore, the BCPNN abstraction constitutes an alternative approach to tackle the energy efficiency wall for brain-sized simulations discussed in Hasler and Marr (2013): Instead of simulating every single neuron and synapse, one can choose a higher level of abstraction for the basic computational units in the brain (e.g., a minicolumn), use a powerful learning rule (e.g., spike-based BCPNN), and implement such networks in a lazy simulation scheme (e.g., on dedicated digital hardware), to finally achieve a very energy-efficient simulation of the brain.

AUTHOR CONTRIBUTIONS

Bernhard Vogginger, René Schüffny, Anders Lansner, Love Cederström, Johannes Partzsch, and Sebastian Höppner designed and conceived this work. René Schüffny and Bernhard Vogginger developed the analytical II solution. Bernhard Vogginger, Love Cederström, Johannes Partzsch, and Anders Lansner provided simulation or analysis code. Bernhard Vogginger, Love Cederström and Johannes Partzsch performed the experiments and analyzed the data. Bernhard Vogginger, René Schüffny, Anders Lansner, Love Cederström, Johannes Partzsch, and Sebastian Höppner wrote the paper and approved the final manuscript.

ACKNOWLEDGMENTS

We would like to thank Édi Kettemann for his work on the analytical I solution. This research was supported by the European Union Seventh Framework Programme (FP7) under Grant Agreement No. 604102 (Human Brain Project), and by the Swedish Science Council (VR-621-2012-3502) and SeRC (Swedish e-Science Research Centre).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: http://www.frontiersin.org/journal/10.3389/fnins.2015.00002/abstract

REFERENCES

Arnold, O., Matus, E., Noethen, B., Winter, M., Limberg, T., and Fettweis, G. (2014). Tomahawk: parallelism and heterogeneity in communications signal processing mpsocs. *ACM Trans. Embedded Comput. Syst.* 13, 107. doi: 10.1145/2517087

Azghadi, M., Iannella, N., Al-Sarawi, S., Indiveri, G., and Abbott, D. (2014). Spike-based synaptic plasticity in silicon: design, implementation, application, and challenges. *Proc. IEEE* 102, 717–737. doi: 10.1109/JPROC.2014.2314454

Bate, A., Lindquist, M., Edwards, I., Olsson, S., Orre, R., Lansner, A., et al. (1998). A Bayesian neural network method for adverse drug reaction signal generation. *Eur. J. Clin. Pharmacol.* 54, 315–321. doi: 10.1007/s002280050466

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Benjaminsson, S., and Lansner, A. (2011). "Extreme scaling of brain simulations on JUGENE," in *Jülich Blue Gene/P Extreme Scaling Workshop* number FZJ-JSC-IB-2011-02, eds B. Mohr and W. Frings (Jülich: Forschungszentrum Jülich, Jülich Supercomputing Centre).

Berthet, P., Hellgren-Kotaleski, J., and Lansner, A. (2012). Action selection performance of a reconfigurable basal ganglia inspired model with Hebbian–Bayesian Go-NoGo connectivity. *Front. Behav. Neurosci.* 6:65. doi: 10.3389/fnbeh.2012.00065

Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472.

Binzegger, T., Douglas, R. J., and Martin, K. A. (2007). Stereotypical bouton clustering of individual neurons in cat primary visual cortex. *J. Neurosci.* 27, 12242–12254. doi: 10.1523/JNEUROSCI.3753-07.2007

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.* 23, 349–398. doi: 10.1007/s10827-007-0038-6

Brette, R. (2006). Exact simulation of integrate-and-fire models with synaptic conductances. *Neural Comput.* 18, 2004–2027. doi: 10.1162/neco.2006.18.8.2004

Cassidy, A. S., Georgiou, J., and Andreou, A. G. (2013). Design of silicon brains in the nano-cmos era: spiking neurons, learning synapses and neural architecture optimization. *Neural Netw.* 45, 4–26. doi: 10.1016/j.neunet.2013.05.011

Diehl, P. U., and Cook, M. (2014). "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware," in *Proceedings of the 2014 International Joint Conference on Neural Networks (IJCNN)* (Beijing).

Farahini, N., Hemani, A., Lansner, A., Clermidy, F., and Svensson, C. (2014). "A scalable custom simulation machine for the Bayesian confidence propagation neural network model of the brain," in *ASP-DAC* (Suntec City), 578–585.

Fiebig, F., and Lansner, A. (2014). Memory consolidation from seconds to weeks: a three-stage neural network model with autonomous reinstatement dynamics. *Front. Comput. Neurosci.* 8:64. doi: 10.3389/fncom.2014.00064

Furber, S., Galluppi, F., Temple, S., and Plana, L. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gerstner, W., and Kistler, W. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, UK: Cambridge University Press.

Gwennup, L. (2011). Adapteva: more flops, less watts: epiphany offers floating-point accelerator for mobile processors. *Microprocess. Rep.* 2, 1–5. Available online at: http://www.adapteva.com/wp-content/uploads/2012/08/adapteva_mpr.pdf

Höppner, S., Shao, C., Eisenreich, H., Ellguth, G., Ander, M., and Schüffny, R. (2012). "A power management architecture for fast per-core DVFS in heterogeneous MPSoCs," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on* (Seoul), 261–264.

Hasler, J., and Marr, B. (2013). Finding a roadmap to achieve large neuromorphic hardware systems. *Front. Neurosci.* 7:118. doi: 10.3389/fnins.2013.00118

Hasler, P., Kozoil, S., Farquhar, E., and Basu, A. (2007). "Transistor channel dendrites implementing hmm classifiers," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on* (New Orleans, LA: IEEE), 3359–3362.

Henker, S., Partzsch, J., and Schüffny, R. (2012). Accuracy evaluation of numerical methods used in state-of-the-art simulators for spiking neural networks. *J. Comput. Neurosci.* 32, 309–326. doi: 10.1007/s10827-011-0353-9

Houzel, J.-C., Milleret, C., and Innocenti, G. (1994). Morphology of callosal axons interconnecting areas 17 and 18 of the cat. *Eur. J. Neurosci.* 6, 898–917. doi: 10.1111/j.1460-9568.1994.tb00585.x

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Johansson, C., and Lansner, A. (2004). *BCPNN Implemented with Fixed-Point Arithmetic.* Technical Report TRITA-NA-P0403, Royal Institute of Technology, Department of Numerical Analysis and Computer Science, Stockholm.

Johansson, C., and Lansner, A. (2007). Towards cortex sized artificial neural systems. *Neural Netw.* 20, 48–61. doi: 10.1016/j.neunet.2006.05.029

Johansson, C., Sandberg, A., and Lansner, A. (2001). *A Capacity Study of a Bayesian Neural Network with Hypercolumns.* Technical Report TRITA-NA-P0120, Royal Institute of Technology, Department of Numerical Analysis and Computer Science. Stockholm.

Johansson, C., Raicevic, P., and Lansner, A. (2003). "Reinforcement learning based on a bayesian confidence propagating neural network," in *SAIS-SSLS Joint Workshop* (Örebro).

Kaplan, B. A., and Lansner, A. (2014). A spiking neural network model of self-organized pattern recognition in the early mammalian olfactory system. *Front. Neural Circuits* 8:5. doi: 10.3389/fncir.2014.00005

Kuhn, A., Aertsen, A., and Rotter, S. (2003). Higher-order statistics of input ensembles and the response of simple model neurons. *Neural Comput.* 15, 67–101. doi: 10.1162/089976603321043702

Lansner, A., and Ekeberg, Ö. (1989). A one-layer feedback artificial neural network with a bayesian learning rule. *Int. J. Neural Syst.* 1, 77–87. doi: 10.1142/S0129065789000499

Lansner, A., and Holst, A. (1996). A higher order Bayesian neural network with spiking units. *Int. J. Neural Syst.* 7, 115–128. doi: 10.1142/S0129065796000816

Lansner, A., Marklund, P., Sikström, S., and Nilsson, L.-G. (2013). Reactivation in working memory: an attractor network model of free recall. *PloS ONE* 8:e73776. doi: 10.1371/journal.pone.0073776

Lansner, A., Hemani, A., and Farahini, N. (2014). "Spiking brain models: computation, memory and communication constraints for custom hardware implementation," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (Suntec City: IEEE), 556–562.

Lansner, A. (2009). Associative memory models: from the cell-assembly theory to biophysically detailed cortex simulations. *Trends Neurosci.* 32, 178–186. doi: 10.1016/j.tins.2008.12.002

Lennie, P. (2003). The cost of cortical computation. *Curr. Biol.* 13, 493–497. doi: 10.1016/S0960-9822(03)00135-0

Lindquist, M., Stahl, M., Bate, A., Edwards, I., and Meyboom, R. (2000). A retrospective evaluation of a data mining approach to aid finding new adverse drug reaction signals in the who international database. *Drug Saf.* 23, 533–542. doi: 10.2165/00002018-200023060-00004

Lundqvist, M., Compte, A., and Lansner, A. (2010). Bistable, irregular firing and population oscillations in a modular attractor memory network. *PLoS Comput. Biol.* 6:e1000803. doi: 10.1371/journal.pcbi.1000803

Lundqvist, M., Herman, P., and Lansner, A. (2011). Theta and gamma power increases and alpha/beta power decreases with memory load in an attractor network model. *J. Cogn. Neurosci.* 23, 3008–3020. doi: 10.1162/jocn/a/00029

Lundqvist, M., Herman, P., and Lansner, A. (2013). Effect of prestimulus alpha power, phase, and synchronization on stimulus detection rates in a biophysical attractor network model. *J. Neurosci.* 33, 11817–11824. doi: 10.1523/JNEUROSCI.5155-12.2013

Meli, C., and Lansner, A. (2013). A modular attractor associative memory with patchy connectivity and weight pruning. *Network* 24, 129–150. doi: 10.3109/0954898X.2013.859323

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybern.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Noack, M., Krause, M., Mayr, C., Partzsch, J., and Schüffny, R. (2014). "VLSI implementation of a conductance-based multi-synapse using switched-capacitor circuits," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on* (Beijing: IEEE), 850–853.

Noethen, B., Arnold, O., Perez Adeva, E., Seifert, T., Fischer, E., Kunze, S., et al. (2014). "A 105GOPS 36mm2 heterogeneous SDR MPSoC with energy-aware dynamic scheduling and iterative detection-decoding for 4G in 65nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International* (San Francisco, CA), 188–189.

Ros, E., Carrillo, R., Ortigosa, E. M., Barbour, B., and Agís, R. (2006). Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics. *Neural Comput.* 18, 2959–2993. doi: 10.1162/neco.2006.18.12.2959

Sandberg, A., Lansner, A., Petersson, K., and Ekeberg, O. (2000). A palimpsest memory based on an incremental Bayesian learning rule. *Neurocomputing* 32-33, 987–994. doi: 10.1016/S0925-2312(00)00270-8

Sandberg, A., Tegnér, J., and Lansner, A. (2003). A working memory model based on fast hebbian learning. *Network* 14, 789–802. doi: 10.1088/0954-898X/14/4/309

Schemmel, J., Grübl, A., Hartmann, S., Kononov, A., Mayr, C., Meier, K., et al. (2012). "Live demonstration: a scaled-down version of the BrainScaleS wafer-scale neuromorphic system," in *Proceedings of the 2012 IEEE International Symposium on Circuits and Systems (ISCAS)* (Seoul), 702.

Seo, J.-S., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., et al. (2011). "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE* (San Jose, CA: IEEE), 1–4.

Silverstein, D. N., and Lansner, A. (2011). Is attentional blink a byproduct of neocortical attractors? *Front. Comput. Neurosci.* 5:13. doi: 10.3389/fncom.2011.00013

Tully, P. J., Hennig, M. H., and Lansner, A. (2014). Synaptic and nonsynaptic plasticity approximating probabilistic inference. *Front. Synaptic Neurosci.* 6:8. doi: 10.3389/fnsyn.2014.00008

Wahlgren, N., and Lansner, A. (2001). Biological evaluation of a hebbian–bayesian learning rule. *Neurocomputing* 38, 433–438. doi: 10.1016/S0925-2312(01)00370-8

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 22 September 2014; accepted: 03 January 2015; published online: 22 January 2015.

*Citation: Vogginger B, Schüffny R, Lansner A, Cederström L, Partzsch J and Höppner S (2015) Reducing the computational footprint for real-time BCPNN learning. Front. Neurosci. **9**:2. doi: 10.3389/fnins.2015.00002*

This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience.