SANDIA REPORT
SAND2020-10377
Printed September 2020

Sandia National Laboratories

Neuromorphic scaling advantages for energy-efficient random walk computations

J. Darby Smith, Aaron J. Hill, Leah E. Reeder, Brian C. Franke, Richard B. Lehoucq, Ojas Parekh, William Severa, and James B. Aimone

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy

Office of Scientific and Technical Information

P.O. Box 62

Oak Ridge, TN 37831

Telephone: (865) 576-8401

Facsimile: (865) 576-5728

E-Mail: reports@osti.gov

Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce

National Technical Information Service

5301 Shawnee Rd

Alexandria, VA 22312

Telephone: (800) 553-6847

Facsimile: (703) 605-6900

E-Mail: orders@ntis.gov

Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

Computing stands to be radically improved by neuromorphic computing (NMC) approaches inspired by the brain's incredible efficiency and capabilities. Most NMC research, which aims to replicate the brain's computational structure and architecture in man-made hardware, has focused on artificial intelligence; however, less explored is whether this brain-inspired hardware can provide value beyond cognitive tasks. We demonstrate that high-degree parallelism and configurability of spiking neuromorphic architectures makes them well-suited to implement random walks via discrete time Markov chains. Such random walks are useful in Monte Carlo methods, which represent a fundamental computational tool for solving a wide range of numerical computing tasks. Additionally, we show how the mathematical basis for a probabilistic solution involving a class of stochastic differential equations can leverage those simulations to provide solutions for a range of broadly applicable computational tasks. Despite being in an early development stage, we find that NMC platforms, at a sufficient scale, can drastically reduce the energy demands of high-performance computing platforms.

ACKNOWLEDGEMENTS

We thank Steve Plimpton and Andrew Baczewski for reviewing an early version of the manuscript detailed in this report. We thank Conrad James, the project PM, for support through this project. We thank Craig Vineyard for managing the neuromorphic computing resources that we were able to leverage in this research.

The authors acknowledge financial support from Sandia National Laboratories' Laboratory Directed Research and Development Program.

This report in its entirety is in parallel being under review for journal publication.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
AI	Artificial Intelligence
DTMC	Discrete Time Markov Chain
LIF	Leaky Integrate-and-Fire
NMC	Neuromorphic Computing
PIDE	Partial Integro-differential Equation
RW	Random Walk
SDE	Stochastic Differential Equation
VN	Von Neumann

1. INTRODUCTION

The efficiency of biological nervous systems has intrigued even the earliest designers of computing systems [22, 36], but the theoretical value of neuromorphic hardware remains unclear. While quantum computing offers clear fundamental advantages at scale [31], the advantages of NMC are more subtle, a fact that has muted enthusiasm despite the increasing ability to develop large scale neural processors today [10, 12, 23]. Nonetheless, in addition to the advanced cognitive capabilities, there are several architectural features of most nervous systems that may yield advantages including the high degree of connectivity between neurons, the colocation of processing and memory, and the use of action potentials (i.e., spikes) to communicate.

Algorithms research for spiking neuromorphic hardware has primarily focused on its suitability for deep learning and other emerging AI algorithms [27, 30]. This application is straight-forward, given the alignment of neural architectures with neural networks, and it can be expected that the value of NMC will grow as AI algorithms derive further inspiration from the brain [2]. However, the impact of NMC beyond cognitive applications is less clear. Quantum computing provides a precedent for emerging hardware to have impact beyond its original inspiration: while quantum computing was conceived as a means for efficient chemistry simulations [11, 20], it is now recognized that it can impact a much broader range of computing applications [5, 17, 31]. Along these lines, there is growing evidence that neuromorphic hardware can provide theoretical complexity advantages on a growing set of non-cognitive, non-AI applications [3, 4, 24, 25, 28]. Unlike quantum computing, which still faces technical challenges in scaling up to sizes necessary for real-world impact (as noted by the recent findings concerning *quantum supremacy* [5]), NMC platforms can already be scaled to non-trivial sizes, with several multi-chip spiking NMC systems achieving scales of over a hundred million neurons. Nevertheless, NMC systems remain smaller and less efficient than the human brain, and the critical scales for NMC remain unknown since the appropriateness of an analogous concept of *neuromorphic supremacy* remains unclear.

Identifying NMC's value for an application is complicated by the fact that its advantage primarily derives from its energy-efficiency as opposed to a promise of faster computation (although speed benefits remain a possibility), and because NMC is an immature technology compared to conventional von Neumann (VN) systems, which have been optimized and advanced over decades in both hardware and software. Here, we define a concept of a *neuromorphic advantage* to describe algorithms for which NMC shows comparable or better time-scaling compared to a VN architecture while requiring less energy (i.e., "energy efficiency") to perform the same computation.

In this report, we identify a neuromorphic advantage for large-scale spiking neuromorphic hardware on a fundamental numerical computing task: solving partial integro-differential equations (PIDEs) that have probabilistic representations involving a jump-diffusion stochastic differential equation (SDE). The solutions to these PIDEs can be approximated by averaging over many independent random walks (RWs), a process often referred to as Monte Carlo. Diffusion is a quintessential component of the underlying SDEs used in the probabilistic solution of the PIDEs. We can show our NMC algorithm for generating RW approximations to diffusion satisfies our neuromorphic advantage criteria on two current large-scale neuromorphic platforms: the IBM Neurosynaptic system [23], also known as TrueNorth and introduced in 2014, and the Intel Loihi system [10],

introduced in 2018. While distinct neural architectures, both directly implement a large number of neurons in silicon (1 million and 128 thousand per chip, respectively), are readily-scalable to multi-chip platforms, and are reflective of the long-term technology trends in spiking neuromorphic hardware. We then show that our NMC algorithm for random walks can be extended to account for more sophisticated jump-diffusion processes that are useful for addressing a wide range of applications, including financial economics (e.g., option pricing models), particle physics (e.g., radiation transport), and machine learning (e.g., diffusion maps).

2. RANDOM WALKS AND SPIKING NEUROMORPHIC HARDWARE

RW solutions are often an attractive option for large scale modeling efforts since independent RWs can readily be computed in parallel. Countering these benefits is the large number of RWs required to approximate solutions via a Monte Carlo method, and translating large-scale RW-based particle codes to GPU-heavy computing platforms is an active area of research [15]. Our approach leverages two key features of spiking neuromorphic hardware – the parallel computation of neurons and the event-driven spiking communication between them – to perform a highly efficient mapping of stochastic processes. While deterministic numerical solutions of PDEs often rely on relatively few large complex calculations, RWs typically rely on many simple computations. As we show, these computations can be efficiently implemented within circuits of spiking neurons.

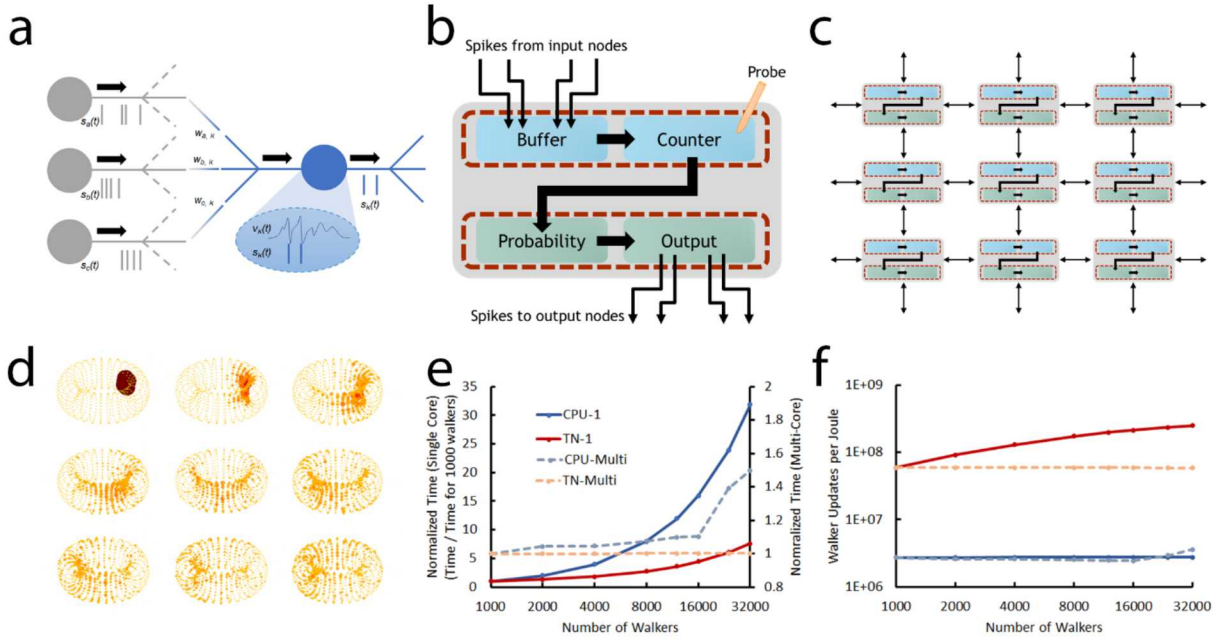


Figure 1: Neuromorphic hardware can efficiently implement random walks. (a) Leaky integrate-and-fire (LIF) neurons on spiking neuromorphic hardware integrate activity from many inputs, generate a 'spike' if an internal threshold is crossed, and only communicate to targets if the spike exists. (b) Random walk transitions can be performed and tracked by a counter circuit combined with a stochastic output. Each circuit typically comprises of between 10 and 20 LIF neurons, depending on the number of edges. (c) Random walk transition circuits are repeated for every mesh point, and the graph of mesh points equates to the state transition matrix of a discrete time Markov chain. The NMC algorithm implements both the stochastic and deterministic state transitions of all random walkers at all mesh points in parallel. (d) Demonstration of simple diffusion on a 30x30 torus on the Intel Loihi platform. Aside from reading out intermediate states for visualization, the entire random walk process was performed within the NMC system. (e) Scaling random walkers on a single neural mesh on IBM TrueNorth shows better time scaling than a single Intel Xeon CPU core, while replicating walkers over multiple meshes or multiple cores shows minimal penalty for paralleling simulation over all available resources on either a NMC or CPU chip. All values are normalized to the respective technologies time required for 1000 walkers on a single core. (f) TrueNorth has a considerably higher energy-efficiency (walker updates per

Joule) than CPUs for both single mesh / core and multiple meshes /cores operation modes. All scaling experiments had 10 replicates with standard errors below 0.5%, so error bars are not shown.

Our neural algorithm for RWs is based on a previously described circuit to model diffusion [29]. Each node consists of a simple neural circuit that uses common leaky-integrate and fire (LIF) neurons (**Fig. 1a**) to count the number of incoming spikes and a circuit to stochastically distribute spikes to output nodes (**Fig. 1b**). These nodes are then assembled into a graph whose edges represent the transition probabilities from one state to another (**Fig. 1c**). The neural algorithm can thereby be generally configured to represent any time-homogeneous Discrete-Time Markov Chain (DTMC). During operation, walkers are initialized as spike inputs to the appropriate starting location mesh point, and once the supervisor circuit initiates the model, the spikes' propagation through this mesh directly reflect the movement of random walks through the corresponding state space. In this respect, the neuromorphic hardware implements both the stochastic components of the RW as well as any deterministic components of a stochastic process update.

The shape of this graph and the output probabilities within each node represent the problem description; for instance, a nearest-neighbor mesh with uniform probabilities would lead to Brownian motion in the limit as the mesh and time step go to zero (**Fig. 1d**). More sophisticated RWs, including those with non-local and jump diffusion, walker absorption and creation, and location-dependent transition probabilities, can readily be implemented in this framework.

We first performed scaling studies to assess the computational costs inherent in simulating RWs on neuromorphic hardware in practice relative to CPUs. We observed that increasing the density of RWs in the NMC implementation required relatively less additional time (sub-linear scaling) compared to linear scaling when increasing walkers on a commodity server-class CPU (Intel Xeon E5-2662; **Fig. 1e**), although the CPU simulates random walks faster than our IBM TrueNorth implementation, whose neurons are designed to match biological speeds (2.5 seconds vs 1350.9 seconds for 1000 walkers over 100,000 timesteps). Likewise, we observed that increasing walker counts by replicating the NMC meshes exhibited similar scaling as spreading RW simulations over multiple cores (**Fig. 1e**). The Loihi platform showed similar scaling trends and faster overall processing than TrueNorth, although overall single-chip capacity was smaller (data not shown).

As NMC systems combine compelling power benefits with slower underlying processing speeds; the important measure of interest is the required total energy (which combines power and speed) to perform equivalent amounts of computational work. As described in **Supplemental Note 1**, seeing competitive time-scaling on a current neuromorphic platform is more informative than the specific processing speed, especially because of the difference in technological maturity between conventional and NMC systems. Despite this maturity difference, TrueNorth already has demonstrated a far lower power consumption (10^{-3} – 10^{-1} W) than high-performance CPUs (~10^{2} W). This several orders-of-magnitude savings in power yields a substantial overall energy-savings even when longer simulation times or increased core usage is considered. While direct measurements of the actual power consumption of computations is not feasible, we indirectly estimated the total energy cost, in Joules based on the observed speed of the computation with the estimated power draw of the hardware. In this measure, which accounts for the speed and power differences between technologies, we observed a rather striking advantage in NMC's energy-efficiency for the simple RW task; showing between a 20x to 100x increase in walker updates per Joule (**Fig. 1f**).

2.1. Neuromorphic Hardware Methods

2.1.1. General

The neurons used on both Loihi and TrueNorth either are integrate-and-fire neurons (IF) or threshold gate (TG) neurons. In both cases, the hardware neurons integrate all active synaptic inputs and make a decision to fire based on a threshold. For IF neurons, if a neuron does not fire, there is no decay of the neuron's internal voltage. In contrast, TG neurons have full decay every time step, regardless of whether it spikes.

For both TrueNorth and Loihi, the implementation of the random walk algorithm was based on the density circuit described in [29]. Each mesh point in the simulation consisted of two counting circuits (one to buffer inputs, one to count down outputs) and a probabilistic fan-out circuit. The network also utilized a population of supervisor neurons to control the timing and synchrony of the walkers through the circuit. For cases where walkers are synchronized (each timestep involves every walker advancing in time by 1), the separate buffer circuit is required to accumulate all walkers coming to a location, however this is unnecessary if the walkers can be run asynchronously. We briefly describe the design of each circuit here in the context of Loihi. The TrueNorth configuration was similar, albeit with some minor differences that are noted.

2.1.2. Buffer and Counter Circuits

In the random walk algorithm, the buffer circuit and walker counting circuit are identical, with the only difference being the inputs and outputs. The counting circuits on the Loihi are structured as shown in Figure 2. Each circuit consists of three neurons: an IF "**count**" neuron, which stores the count of random walkers at that location in its internal voltage (as a negative distance from threshold), a TG "**generator**" neuron, which is designed to spike until the counter neuron reaches its threshold, and a TG "**relay**" neuron, which corrects for situations where there are no walkers at that location. In the buffer circuit, the **count** neuron receives synaptic inputs (weight $=-1$) from other mesh node outputs. In the counter circuit, this neuron receives a synaptic input (weight $=-1$) from the buffer **generator** neuron. In both cases, the **count** neuron will represent the cumulative walker density at that location. The output of the **count** neuron is an inhibitory connection to its respective **generator** neuron, designed to stop its activity

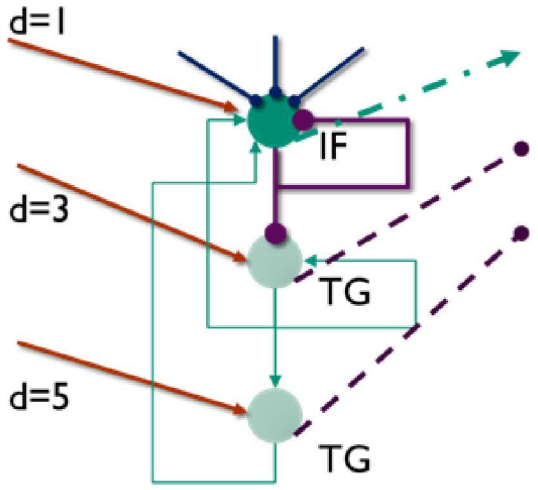


Figure 2: Neural Circuits for Buffering and Counting on Loihi. Red input lines (from left) represent inputs from supervisor neuron. Circle ends represent inhibitory connections (weight = -1), arrows represent excitatory connections (weight = 1). For buffer circuit, outputs (to right) go to counter circuit count neuron; for counter circuit, outputs go to probability neurons.

The circuit is designed so that when the supervisor activates the circuit, the **generator** neuron will continue to spike until the **count** neuron reaches a threshold and sends an inhibitory spike to stop it from firing. Thus, at each simulation timestep, if there are k walkers at a location, the **generator** neuron will fire $k+1$ times (which is subsequently corrected for). During the first half of the simulation timestep, the buffer **generator** neuron transfers the count from the buffer to the counter circuit, and during the second half of the simulation timestep, the counter **generator** neuron transfers the count to the probabilistic fan-out circuit (described below) which then distributes the walkers to a different mesh node's buffer circuit.

The **relay** neuron is present to account for the subtle timing between the **generator** and **count** neurons that provides the extra signal from the **generator** neuron as well as help handle cases where the mesh point has no walkers. There are several mechanisms for performing these corrections, which also result in a few additional synaptic connections, for which the Loihi strategy is shown in Figure 2.

2.1.3. Probability Circuit for Loihi

The goal of the probability circuit is to send a walker, as a spike, to one of the mesh node's downstream target's buffer **count** neuron. The circuit is designed to use intrinsic pseudo-random number generators (PRNGs) available to each individual neuron to select ***only one*** of the mesh node's outputs at the appropriate Markov transition probability.

While there are likely several implementations of a circuit to send a spike to one of N outputs with a probability $p_{out,\, i=1,\ldots,N}$, the methods we selected on Loihi and TrueNorth were identified to account for the particular nature of the PRNGs on each chip.

For Loihi, the PRNG provides a random input onto each neuron on a particular neural core as an 8-bit pseudo-random integer, with potential multiplicative and additive scaling. For our purposes here, we can consider this random number as an integer uniformly chosen from $[-127,128]$. Suppose we want a neuron, b, that has a threshold 100, to fire at probability p when it receives a spike from an upstream neuron, a. We then can set an input weight from a to b as

$w_{ab} = 128 \times p_b + 36.$

To output to one of N neurons, we considered the neural circuit that collapses a probability tree into a single layer. For instance, one can consider a uniform 3 layer decision tree, with each branch having a 50% probability, leading to eight outputs with 12.5% probability. This scales as a depth $log_2(N)$ circuit with $N-1$ total probability neurons. We can compress this into a single layer, by having each output node (the leaves on the decision tree) requiring that all positive input branches are active and receiving inhibition from all "wrong" decision branches, and no inputs from branches on the other side of the decision tree.

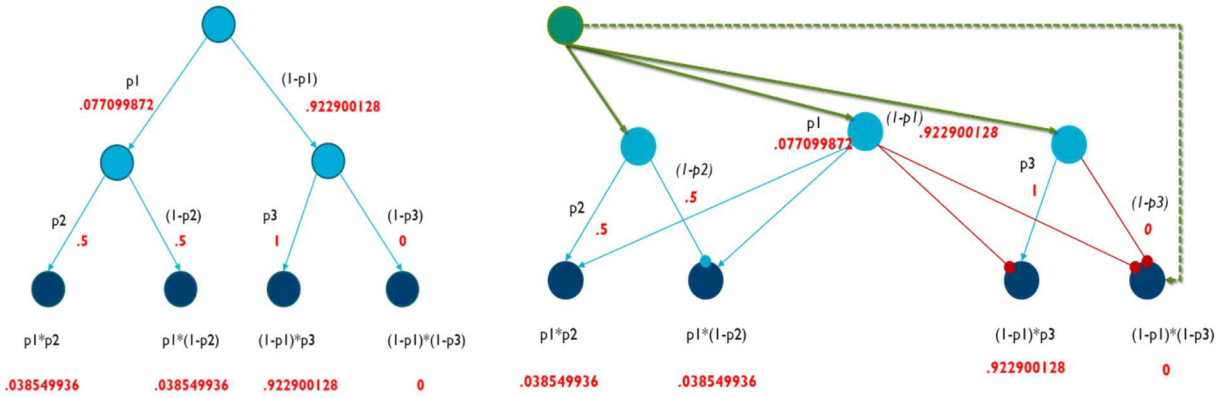


Figure 3: Illustration of computing probabilistic circuit. Left: hypothetical decision tree to compute probabilities with example output probabilities in red. Right: same decision tree compressed into a single layer, with source input driving probabilistic choice. The dotted line is an excitatory connection with a delay to correspond to skipping the probabilistic layer. From source neuron, weights from source neuron (green) to probability neurons (blue) are set to tune probabilities neurons fire, per equation M.1. Outputs of probability neurons with arrows are excitatory (weight = 1) and with circles are inhibitory (weight = -1).

Procedurally, this is achieved by having $N-1$ probability neurons, whose probabilities of being active on a given timestep are given by the following procedure:

Algorithm 1: Determine Probability Circuit

Input: Neuron g that generates a spike per walker

Input: Output nodes $o_1 \ldots o_N$ with desired output probabilities $\nu_1 \ldots \nu_N$

// Assume N is a power of 2

T := A binary tree with $o_1 \ldots o_N$ as the leaf nodes

for *non-leaf node* $n = 1 \ldots N-1$ **do**

  Label one edge as 'positive' and the other as 'negative' **end**

for *non-leaf node* $n = 1 \ldots N-1$ **do** $\Lambda := \{i : o_i$ is a leaf of $n\}$

  $\Lambda_p := \{i :$

  o_i is a leaf of n connected through the 'positive' branch$\}$ $p_n \leftarrow {}^{\mathrm{P}}{}_{i \in \Lambda_p} \nu_i / {}^{\mathrm{P}}{}_{i \in \Lambda} \nu_j$ // Assign a conditional probability to each node

end

// T is a "probability tree" $T\hat{}$:= set of $2N-1$ neurons **for** *node n in T* **do**

  $n\hat{} \in T$:= a neuron with threshold 1 and decay 0 4

  if *n is a leaf node* **then**

    P = $(n_i, e_i)_{1 \ldots L}$:= the unique path from the root node to n c := number of 'positive' edges in P **for** $i = 1 \ldots L$ **do**

      if e_i *is a 'positive' edge* **then**

        $weight[\hat{n}_i, \hat{n}] \leftarrow 1/c + \epsilon$ // All 'positive' edges must have spikes

We then set the threshold to number of positive inputs onto the output neurons. For neurons that don't have any positive inputs, we include a connection with delay = 3 from the spike **generator**.

The output neurons, when activated, will send a single spike to their corresponding mesh node's buffer **count** neuron. This corresponds to the transfer of that walker to the different mesh location.

2.1.4. Probability Circuit for TrueNorth

The probability circuits for Loihi and TrueNorth are functionally equivalent, but due to differences in the two hardware platforms the realizable TrueNorth circuit differs slightly from that of Loihi. It is important to provide adequate coverage of the TrueNorth probability circuit in this section for completeness.

TrueNorth has a few different stochastic neuron dynamics that can be configured for each neuron. For this material we are taking advantage of stochastic leak. The two pertinent neuron equations that emerge when utilizing TrueNorth's stochastic leak function is the leak equation defined as

$$V_j^*(t) = V_j(t) + F\left(\lambda_j, \rho_j^\lambda\right), where,$$
$$F\left(\lambda_j, \rho_j^\lambda\right) = \begin{cases} 1 & if\ \lambda_j \geq \rho_j^\lambda \\ 0 & otherwise \end{cases}.$$

Further, ρ_j^λ is a random sample from the PRNG drawn from $U(0,255)$ and λ_j is an integer in the range of $[0,255]$. The stochastic property of the neuron is ultimately controlled by setting the neuron parameter λ_j. We then set the threshold of this neuron to 1 with a reset potential of 0. Thus, if F evaluates to 1 the neuron will fire, and if F evaluates to 0 the neuron will not fire. In effect, we created a stochastic neuron that fires with some probability defined by F.

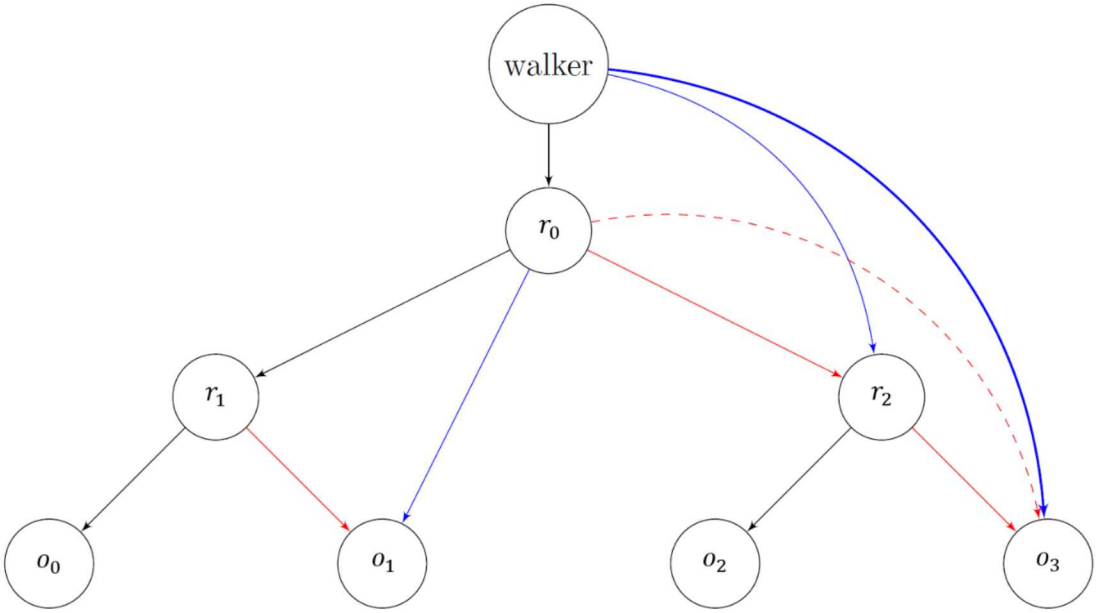


Figure 4: Binary tree representing the stochastic walk through a TrueNorth mesh node. Probability neurons are r_0, r_1, and r_2. Black edges are excitatory, red edges are inhibitory. Blue edges indicate a delay of 1, and bold blue and red dashed edges indicate a delay of 2. The four leaf nodes, o_0, o_1, o_2, and o_3, are the directional nodes with derived exit probabilities.

We exploit this stochastic neuron dynamic to provide a neuron circuit that directs a walker (spike) through a binary tree to give rise to a controllable exit probability for each of the four exit neurons of the mesh node. We define a probability neuron that receives the walker input (spike) but also receives input from the stochastic neuron. We set this neuron's threshold at 1 with a leak value of -1. These probability neurons are depicted in Figure 4 as r_0, r_1, and r_2. Thus, if a walker enters this probability neuron it contributes a value of 1 to the neuron potential and if the stochastic neuron also provides a spike, or value of 1, the potential will have a value of 2. Then, the probability neuron will leak a value of -1 to result in a final potential value of 1 that is compared to the threshold value

of 1, which will result in a fire of that neuron. This neural dynamic requires two spikes to fire, if it only gets 1 spike, meaning the stochastic neuron did not fire, the probability neuron will not fire.

This is the exact formulation that defines the probabilistic nature of the mesh nodes in TrueNorth. The leaf nodes of the tree define the exit directions of each mesh node. An easy way to understand how the three probability nodes influence the exit direction is to consider the combinatorial effects of the probability nodes firing. For o_0 to fire it means r_0 and r_1 fired. For o_1 to fire it means r_0 fired and r_1 did not. For o_2 to fire it means r_0 did not fire and r_2 did. For o_3 to fire it means both r_0 and r_2 did not fire.

More formally, each leaf node's probability of firing can be related to the probabilities of the tree nodes probability to fire. Specifically,

$$\begin{aligned} P(o_0) &= P(r_0) \times P(r_1), \\ P(o_1) &= P(r_0) \times (1 - P(r_1)), \\ P(o_2) &= P(r_2) \times (1 - P(r_0)), \text{ and} \\ P(o_3) &= 1 - (P(r_0) \times (1 - P(r_2)) + P(r_2). \end{aligned}$$

But, in practice we prefer to define the probabilities of each exit direction first and formulate what the stochastic parameter of the stochastic neuron should be. To this end,

$$\begin{aligned} P(r_0) &= P(o_0) + P(o_1), \\ P(r_1) &= \frac{P(o_0)}{P(o_0) + P(o_1)}, \text{ and} \\ P(r_2) &= \frac{P(o_3)}{1 - P(o_0) + P(o_1)}. \end{aligned}$$

On TrueNorth, this stochastic dynamic utilizes, effectively an 8-bit PRNG. Technically, the PRNG on TrueNorth is 32-bits, but the stochastic leak use case masks off the upper 24 bits and delivers the lower 8 bits of the PRNG to the computational circuit of the neuron. From the analytical side of solving PDEs with random walk algorithms the exit probabilities of each mesh node are defined by real numbers in the range of $[0,1]$. To convert the real valued representation to the lower bit resolution, for acceptance into a TrueNorth model, we multiply the real value by 256 and then apply a standard round operation to the result. This essentially scales the range of $[0,1]$ to the range of integers from $[0,256]$. Because the range of λ_j is $[0,255]$, we then shift the scaled range left by 1, effectively creating represented probabilities in the range of $[-1,255]$. What is meant by the value of -1 is a probability of 0. Since -1 is not an acceptable value for the TrueNorth model we handle this by removing the stochastic neurons synapse to the probability neuron and setting the stochastic neuron's stochastic parameter to 0. The removal of that synapse effectively provides it a 0 probability of firing. More specifically, there is a 0 probability of the stochastic neuron delivering a spike to the probability neuron.

2.1.5. Implementing Models on Loihi and TrueNorth

Models are initialized on Loihi and TrueNorth by generating a mesh of equivalent buffer and counter circuits and probabilistic circuits tailored to the outputs defined in the application's Markov Transition table. Specific to TrueNorth, a connectivity diagram is represented in Figure 5 which provides the implementation details of neuron connectivity for implementing the mesh node in

TrueNorth; this is the TrueNorth model representation of Figure 4. Importantly, different mesh points can have distinct numbers of outputs, though more outputs will directly equate to more neurons required. Further, there is no real restriction on the types of graphs and the connectivity, though there will be resource constraints in terms of overall neuron counts and hardware-specific fan-in and fan-out constraints, if any. It should be noted that cases where network connectivity restrictions are problematic can typically be resolved by replicating target neurons or mesh points.

For most of the examples in the main text, the Loihi and TrueNorth neuromorphic models are unchanged, with the only variable input being the transition matrix used to define the connections between mesh nodes and the weights onto the probability neurons in the mesh points. In most examples, inputs are given by providing a sequential number of spikes to the appropriate buffer **count** neuron of the mesh location from which the random walks will be initialized. There are precision considerations in the internal voltage levels that differ between platforms, so we held the number of walkers on Loihi to a maximum of about 1000 for the initial condition; although this likely can be higher. On TrueNorth, the range of voltage potentials can support up to 393215 walkers. Full details of generating Markov transition matrices for the examples in the main text used on Loihi and TrueNorth are given in **Supplemental Note 3**.

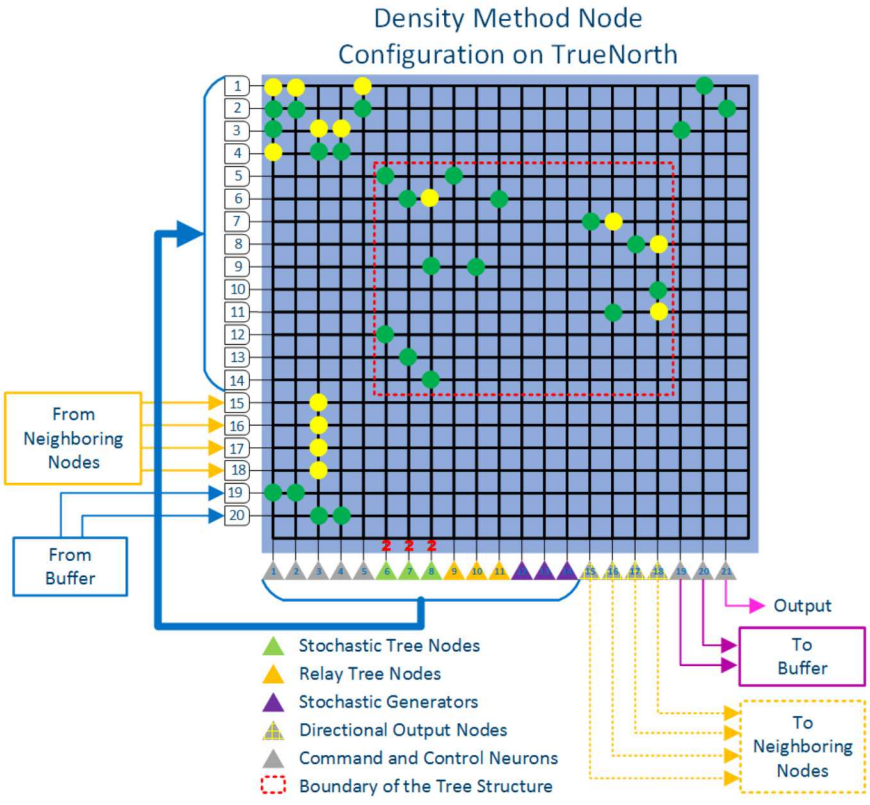


Figure 5: A near complete specification of the TrueNorth mesh node model for a random walk algorithm. This is a more defined representation of the binary tree from Figure 4. Neurons are represented by triangles, neuron inputs are on the left edge of the square and a synapse to a neuron is defined by a circle on the cross bar. Green circles are excitatory connections and yellow circles are inhibitory connections. The red number 2 above neurons 6, 7, and 8 indicate that they

fire as a result of 2 or more incoming spikes, all other neurons fire as a result of 1 or more incoming spike

2.1.6. TrueNorth Scaling Studies: Execution and Statistics

We performed a number of scaling experiments on IBM's Neurosynaptic System (TrueNorth) to better understand how the random walk algorithm performs on actual neuromorphic hardware. The base experiment is a random walk simulation on a 21×21 node torus mesh, where each node has 4 incoming and 4 outgoing connections. This corresponds to a walker on the surface moving up, down, left, or right. The transition probabilities of the 4 directions are all an equal 0.25. All walkers begin the simulation at the center of the mesh, and the simulations are ran for 100,000 time steps. Here, a time step is the unit of time for all walkers to have moved one position.

Initially, the number of walkers is increased from 1000 to 32000 (see Figure 1**e-f** in the main text, red line) in 8 different runs. Then, this is repeated, increasing the number of walkers through parallelization. Copies are made of the underlying mesh, and each mesh is given 1000 random walkers. The mesh copies increase to keep the total walkers in the simulation consistent with the static mesh runs.

Each experiment had 8 parameter sets that scaled the number of walkers or a parameter of parallelism (mesh copies or number of starting locations). Within each of these parameter sets, 10 trial iterations were executed to allow for a meaningfully computation of the mean and standard deviation of the trial set. For execution on TrueNorth, the total number of neural time steps, or ticks, must be defined *a priori*. Because of the random nature of these experiments, achieving exactly $100{,}000$ timesteps is not possible given a fixed number of ticks. Therefore, sample trials were run to achieve a close enough tick count per each parameter set to achieve close to $100{,}000$ timesteps. Then the data was normalized to exactly $100{,}000$ timesteps by computing the tail timestep to tick count ratio. The tail step ratio is defined by taking the mean step ratio of all simulation steps ignoring the first $1{,}000$ simulation steps (this is an arbitrary choice, but, based on empirical evidence, the step ratio flattens out after the first few hundred time steps as the random walk reaches sufficient diffusion). This mean step ratio was then used in conjunction with the tick rate (seconds per tick) to subtract or add time to the measured time of the experiment's trials. For example, if a particular trial ran for $100{,}034$ timesteps, had a mean tail ratio of 17.382, an execution time of 937s, and a tick rate of 0.5ms/tick we would take $937 - 34 \times 17.382 \times 0.0005 = 936.705$s as our normalized execution time.

2.2. Scaling Results

Table 1: Simulation time results of scaling random walks on CPU and TrueNorth. Time measured in seconds for different number of walkers (rows) over 100,000 walker updates. Each simulation was repeated 10 times, errors reported as standard deviations. Left two columns: scaling study using only one CPU core or one mesh on TrueNorth (which uses 51 TrueNorth neural cores). Right four columns: scaling study using increasing number of CPU cores and TrueNorth meshes.

Walker Total	CPU-1	TN-1	CPU-Cores	CUP-Multi	TN-Cores	TN-Multi

1000	2.55 ± 0.05	1350.85 ± 3.04	1	2.58 ± 0.04	51	1350.79 ± 1.99
2000	5.11 ± 0.11	1757.66 ± 4.16	2	2.69 ± 0.01	102	1350.39 ± 2.29
4000	10.16 ± 0.05	2461.87 ± 2.65	4	2.69 ± 0.02	204	1349.61 ± 2.37
8000	20.29 ± 0.03	3716.55 ± 3.51	8	2.77 ± 0.05	408	1351.28 ± 2.43
12000	30.43 ± 0.05	4885.58 ± 3.72	12	2.83 ± 0.03	612	1352.02 ± 1.60
16000	40.62 ± 0.11	6009.70 ± 3.59	16	2.84 ± 0.03	816	1352.03 ± 0.35
24000	60.93 ± 0.14	8181.83 ± 3.08	24	3.58 ± 0.04	1224	1352.49 ± 0.36
32000	81.20 ± 0.15	10292.55 ± 4.12	32	3.86 ± 0.04	1632	1352.49 ± 2.89

Table 2: Energy results of scaling random walks on CPU and TrueNorth. Left: Results in [table:scaling] converted to walker updates per second. Right: Energy estimates provided by [eq:energy_neural] and [eq:energy_vn], Power consumption was taken published values of upper bound estimated power consumption for each chip (115W TDP for Xeon CPU, 100mW for IBM TrueNorth) proportioned to core usage.

Walker Total	**CPU-1**	**TN-1**	**CPU-Multi**	**TN-Multi**	**CPU-1**	**TN-1**	**CPU-Multi**	**TN-Multi**
1000	3.92E+02	7.40E+04	3.88E+07	7.40E+04	2.73E+06	5.95E+07	2.70E+06	5.95E+07
2000	3.92E+07	1.14E+05	7.44E+07	1.48E+05	2.72E+06	9.14E+07	2.59E+06	5.95E+07
4000	3.94E+07	1.62E+05	1.49E+08	2.96E+05	2.74E+06	1.30E+08	2.58E+06	5.95E+07
8000	3.94E+07	2.15E+05	2.89E+08	5.92E+05	2.74E+06	1.73E+08	2.51E+06	5.94E+07
12000	3.94E+07	2.46E+05	4.24E+08	8.88E+05	2.74E+06	1.97E+08	2.46E+06	5.94E+07
16000	3.94E+07	2.66E+05	5.63E+08	1.18E+06	2.74E+06	2.14E+08	2.45E+06	5.94E+07
24000	3.94E+07	2.93E+05	6.70E+08	1.77E+06	2.74E+06	2.36E+08	2.91E+06	5.94E+07
32000	3.94E+07	3.11E+05	8.29E+08	2.37E+06	2.74E+06	2.50E+08	3.60E+06	5.94E+07

Table 3: 4000 walkers are initialized in the center of our 21×21 torus mesh, distributed equally across various copies of the mesh. The average execution time normalized to 100,000 time steps on TrueNorth is given from 10 trials each.

Copies	Average Time (S)	Std. Dev.	Cores
1	2462.19	1.33	51
2	1757.19	2.79	102
4	1350.27	1.61	204
8	1108.28	1.94	408
16	953.29	3.82	816
32	857.51	2.05	1632
40	830.06	2.31	2040
50	809.74	3.53	2550

Table 4: The given number of walkers are initialized in the center of a single copy of our 21×21 torus mesh. The transition probabilities to the four possible transition locations are randomized. The average execution time normalized to 100,000 time steps on TrueNorth is given from 10 trials each.

Walker Total	Average Time (S)	Std. Dev.
1000	2834.90	860.74
2000	5107.82	2039.43
4000	9850.43	3405.16
8000	19683.49	9327.33
12000	23364.78	8284.22
16000	36347.31	12414.64
24000	59260.23	28048.49
32000	84120.91	32528.64

Table 5: 4000 walkers are divided equally among the given number of starting positions, chosen randomly, on a single copy of our 21×21 torus mesh. The average execution time normalized to 100,000 time steps on TrueNorth is given from 10 trials each.

Starting Positions	Average Time (S)	Std. Dev.
1	2461.74	2.77
4	899.96	123.47
16	933.00	99.88
40	914.15	89.62
80	887.72	97.05
125	884.36	101.70
200	916.44	79.50
400	951.23	86.97

2.2.1. Additional Scaling Studies on TrueNorth

In addition to the previously described staling studies, we also performed 3 additional studies evaluating the random walk algorithm on TrueNorth. In our third experiment, the total count of walkers is held fixed at 4000, but multiple copies of the mesh are instantiated, and the 4000 walkers are equally distributed over all the copies. The fourth experiment is the same as our initial scaling study on a static mesh, however every node is assigned a random transition probability. These are drawn from a uniform distribution, and the sum of all transition probabilities is forced to sum to 1. The final experiment increases walkers by maintaining a single copy of the mesh, but distributing each additional set of 4000 random walkers over multiple starting locations, chosen at random from a uniform distribution.

Experiment 3 further explores the nature of parallelizing the random walk by holding the number of walkers constant at 4000 and distributing them equally over more and more mesh copies. That is, by the last data point of Figure 6, at 50 copies of the mesh there are 80 walkers present on each mesh. As can be seen in figure 3 we achieve a significant reduction in execution time with diminishing returns beginning at 16 mesh copies. This is specific to the mesh size used in this experiment, however. With a 21×21 mesh size there are 441 nodes. Therefore, as the walkers spread out if there are less walkers than total nodes then the mean spread is tending towards one walker per node which is the lower bound for execution time in terms of the time step to tick ratio. At 16 mesh copies and 4000 walkers there are 250 walkers per mesh copy. Fewer walkers on the mesh will result in a quicker spread to the lower bound of execution time of the algorithm. The results of Experiment 3 are displayed graphically in Figure [fig:tn_ex3], while the data points are in Table [table:parallel_times_unif].

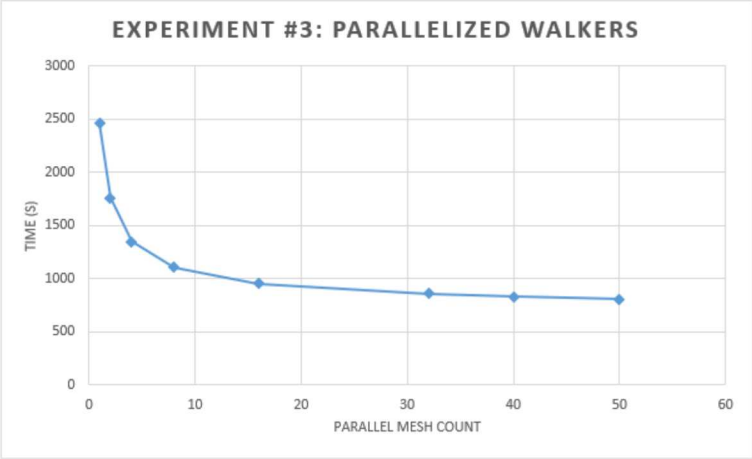


Figure 6: Results of TrueNorth Experiment #3. Execution time reaches a limit as mesh counts increase.

Experiment 4 demonstrates the random walk algorithm's sensitivity to transition probabilities. In this experiment the transition probabilities between nodes where randomized. Figure 7 shows that we still achieve linear scaling as the number of walkers increase, but the error bars grow significantly as the mesh becomes crowded with walkers. The error bars are defined as 1 standard deviation of the experiment's trial set. These results show the sensitivity of the algorithm to execution time bottle necks due to transition probabilities that define areas of the mesh that have high likelihood of walkers entering but low likelihood of them leaving. Such a situation would cause the mean step ratio to be very high, lengthening the number of ticks required to reach 100,000 simulation steps. The results of Experiment 4 are plotted in Figure 7 and data points are available in Table 4.

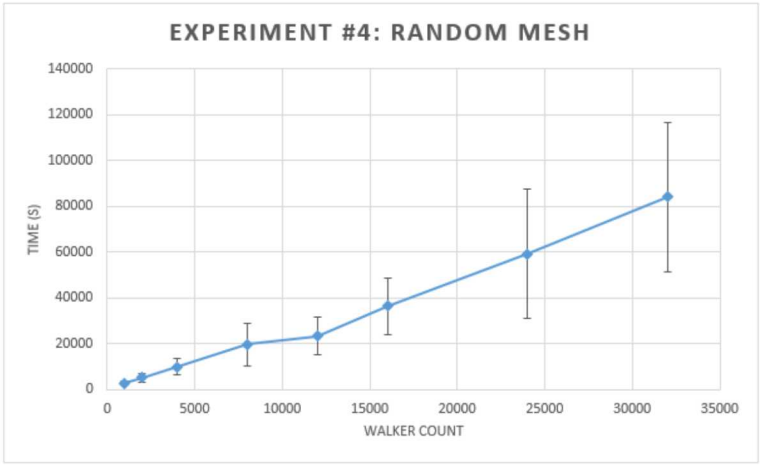


Figure 7: Results of TrueNorth Experiment #4. Execution time scales linearly with walker count, again, but also demonstrates the sensitivity of the algorithm to bottlenecks caused by uneven transition probabilities.

Experiment 5 demonstrates the effect of providing an *a priori* spread of the walkers. From Figure 8 we see that a slight initial spread has a large effect on reducing execution time. We also reach diminishing returns very quickly. The initial data point is 4000 walkers all starting at one node on the mesh, which is consistent with all other experiments. The second data point is putting 1000 walkers each at 4 random locations and we see that any further initial spread does not make a significant difference in execution time. It is also interesting to point out that the standard deviation of the trial sets are noticeable, on the order of 10% of the mean. This is because some random choices could clump walker starting nodes together which would then have a greater likelihood of moving back and forth to each other, slowing the diffusion. Whereas starting locations that are maximally separated in the mesh will diffuse faster and thus execute faster. The results of this final experiment are displayed in Figure 8 and data are given in Table 5.

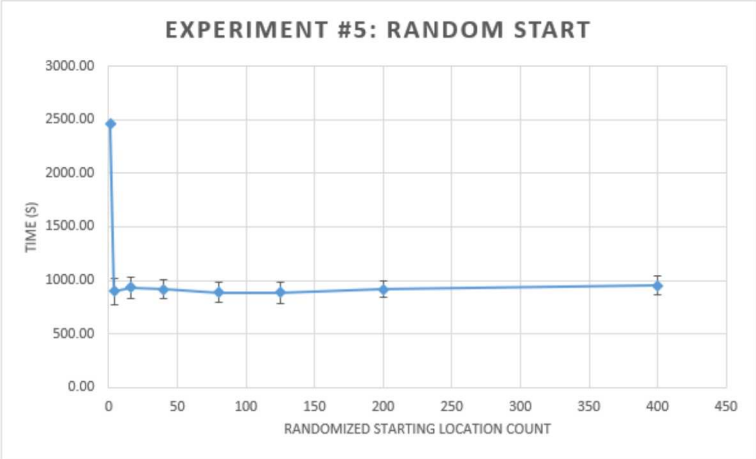


Figure 8: Results of TrueNorth Experiment #5. Execution time is dramatically reduced once all walkers do not start on the same position.

2.3. Supplementary Note 1: Computational Complexity

There is no fundamental reason to expect neuromorphic computing to belong to a different computational complexity class (such as allowing exponential algorithms to be computed in polynomial time; as can be the case with quantum computing) than von Neumann architectures, as proposed neuromorphic systems largely leverage the same physics. However, several aspects of neuromorphic computing can be formally shown to provide scaling benefits. Specifically, neuromorphic hardware can be shown to provide formal advantages due to both reduced communication inherent in processing-in-memory [1], and high neuron fan-in/fan-out ([25, 32].

In this paper, we define a neuromorphic advantage as an algorithm that shows a demonstrable advantage in one resource (e.g., energy) while exhibiting comparable scaling in other resources (e.g., time).

Unfortunately, making apples-to-apples comparisons of an NMC algorithm to a VN algorithm is non-trivial, because while the objective may be the same, the respective algorithms should be tailored to the underlying architecture (stated differently, our spiking algorithm would be an inefficient approach to compute random walks on standard hardware, and neuromorphic hardware

cannot directly implement a conventional implementation). For this reason, we compare our NMC algorithm to the most straightforward VN algorithm, even though these implementations may be formulated quite differently. This analysis is relatively straightforward for the case of a simple Markovian random walk, which we explore here.

2.3.1. Objective

For this analysis, we consider a simple Markovian random walk model over a pre-defined state space of size K, which for most applications we will benefit from increasing the number of walkers to the greatest extent permitted by computational resources. While the approach described in our paper can extend to more complex physics, such as particle absorption, we focus on the simplest scenario here.

For our analysis, we will consider the time (T) and energy (E) scaling of independently simulating a number of walkers, W, over S time steps.

To simplify the analysis, we focus our analysis on a hypothetical single-chip system, though the analysis extends to multi-chip systems insofar as chip to chip communication is similar between architectures. We further focus exclusively on the DTMC simulation itself, as opposed to any averaging or other subsequent analysis of the outputs of the runs.

2.3.2. Complexity of Random Walks on a Parallel von Neumann System

We consider specifically the case of a system with multiple von Neumann processors in parallel, such as a CPU with multiple cores. We define P as the number of processors, or programmable cores, on the chip. In the absence of interactions, the most efficient straightforward implementation of a large number of walkers would be to distribute the walkers evenly over P and walkers would persist on that core through the length of the simulation. Under these conditions, the time of the simulation, T_{VN}, would be

$$T_{VN} = C_{VN,\,time}\frac{W \times S}{P},$$

where $C_{VN,\,time}$ is a constant describing the hardware-dependent time cost of the RW operations on the von Neumann architecture. Likewise, the total energy required to perform the DTMC simulation would be

$$E_{VN} = C_{VN,\,energy}(W \times S),$$

where $C_{VN,\,energy}$ is a constant describing the hardware-dependent energy cost of the RW operations.

From these equations, it can be seen that increasing the density of cores on a chip (or number of chips) can speed up a simulation dramatically, however it does not yield any power savings, which remains proportional to the total computational work of the task.

2.3.3. Complexity of Random Walks on NMC

The NMC algorithm that we consider is fundamentally different than the straightforward VN algorithm. While it is possible to dedicate a subset of neurons to model each walker (see the particle method in [29]), the approach explored here uses neurons to explicitly model the state space over

which walkers may randomly walk as a graph, with a small circuit of neurons at each mesh point. At each simulation time, the circuit at each mesh point distributes its walkers through its edges by the probabilities defined in the Markovian transition matrix. This method, which is referred to as the density method in [29] since the algorithm directly represents the density of walkers at each timestep, has a time cost given by

$$T_{neural} = c_{neural,\,time}\frac{W \times S}{M},$$

where $c_{neural,\,time}$ is the corresponding time-cost of updating a single walker's position on the mesh for one timestep.

More complicated, however, is the denominator M, which is the neuromorphic analogue to the number of cores, P, on a VN processor. M both captures the inherent parallelism of neurons on a chip, which varies across architectures, as well as the distribution of walkers over the neurons, which is a non-deterministic characteristic.

In an ideal NMC system, where every neuron is computed truly in parallel, M would approach a maximum value, M_{max}, which is the number of mesh-points K, in the state space. Currently, most neuromorphic architectures leverage cores that are responsible for computing a subset of neurons, making computation across cores fully parallel with differing degrees of parallelism within cores. So conservatively we can also limit M_{max} to no greater than the number of distinct neural cores N_{cores} on a chip.

In addition, M will vary with walker distribution over time. If walker updates are synchronized, then the DTMC simulation can only advance forward a timestep after all walkers have been processed at each mesh node. If we consider an initial condition where all walkers start at the same mesh point, it will take W hardware time steps for the model to progress one time step; however, if the model is fully mixed (walkers distributed evenly over the mesh), then the time to simulate one time step drops considerably.

In practice, this will make M highly dependent on both the physics being modeled (how fast do the walkers distribute over the mesh), the ratio of number of walkers to the mesh-size, and the relevant time of simulation (longer simulations permit more mixing and thus more time-efficient timesteps). Further, in practice one could replicate mesh-points which can be anticipated as chokepoints due to initial conditions (such as parallelizing start nodes), but this is not always foreseeable in simulations.

For our purposes, we consider that M will eventually approach the minimum of the number of mesh points and number of independent neural cores for long simulations. That is,

$$M \approx min(N_{cores}, K).$$

However, as with the conventional processing, the total energy consumption should be independent of M:

$$E_{neural} = c_{neural,\,energy}(W \times S).$$

2.3.4. Identifying a Neuromorphic Advantage

Per our definition above, our approach can show a neuromorphic advantage is if the time-scaling of the algorithm on neuromorphic hardware is preserved while showing an energy advantage. While we know neuromorphic hardware shows an absolute power advantage (typical neuromorphic chips have sub-1W power requirements vs 100W for server-class CPUs), a power advantage can be easily offset if the computation takes much longer to complete.

We performed scaling studies on a single TrueNorth chip and a commodity class dual CPU system, using Intel Xeon E5-2665, which has 8 CPU cores capable of 16 threads. While both platforms are several years old at this point, they are representative of the state-of-the-art in process engineering at the time of their development. We programmed each platform to implement a simple random walk over a small mesh (21×21 spatial grid, configured as a torus) using a platform-appropriate algorithm. The implementation was using C++ with OpenMP to leverage multiple threads on the Xeon CPUs and MATLAB corelets for the TrueNorth implementation.

We measured scaling in two ways. First, we measured model scaling on a single core / NMC mesh by progressively increasing the number of walkers on a single core / mesh. Second, we performed a standard "weak scaling" experiment; increasing the number of walkers along with the number of cores / NMC meshes. Each simulation ran for 100,000 walker updates with walker counts increasing through [1000, 2000, 4000, 8000, 12000, 16000, 24000, 32000]. Through the C++ code on the CPUs or MATLAB scripts for TN, we measured only the simulation time, ignoring all costs associated with model setup or averaging over the walkers. For each of these simulations, we performed 10 replicate runs, though the variability across runs was typically negligible.

From these scaling studies, we first measured the time to complete these simulations (Table 1). As expected, the CPU implementation is considerably faster than the neuromorphic simulation for all scenarios tested. However, we do see that the ability of a single NMC mesh to distribute additional walkers provides NMC with an advantage in terms of time-scaling, suggesting that a single NMC mesh scales at the rate of multiple CPU cores. The weak scaling experiment tracks a different form of scaling, whereby we use multiple CPU cores / NMC meshes to simulate walkers in parallel. Here, we observe that both the CPU and NMC implementation scale similarly, at least up until the CPU starts using multi-threading on cores. *Combined, these results confirm that the scaling of our NMC algorithm on NMC architectures scales similarly, and possibly slightly better, to the standard CPU algorithm on CPUs.*

From these time measurements, based on [eq:time_vn] and [eq:time_neural] above, we approximated the quantities $^{P}/_{c_{VN,\,time}}$ and $^{M}/_{c_{neural,\,time}}$, since the meaning of M is poorly defined. In both cases, these quantities can be measured in units of ***walker updates per second***. Similarly, from the number of CPU and NMC cores used and published estimates of maximum power consumption of these chips (("Intel Xeon Processor E5-2665," n.d.) and [23]), we can roughly estimate the power consumption in watts, or joulse per second. Combining these two measures, we can then approximate the number of ***walker updates per joule*** (Table 2).

This energy-centered analysis shows that despite being considerably slower than conventional CPUs, NMC platforms can be considerably more energy-efficient than von Neumann processors. Not surprisingly, regardless of on-chip parallelization, CPU energy scaling is relatively constant at between 2.5 and 3 million walker updates per Joule. Based on core-utilization estimates, the NMC is considerably more efficient and becomes

more so at larger simulation sizes, starting at about 60 million walker updates per Joule and progressing up to as high as 250 million walker updates per Joule when walker density is high.

It is important to acknowledge that this is an indirect measure of power consumption. For instance, the CPU power measurement used above was based on linearly estimating power requirements based on core usage relative to published values of Total Draw Power (TDP) for a maximum load. Likewise, on event-driven NMC platforms, such as TrueNorth, the actual power draw should be a function of spiking activity, not core usage. It therefore would not be surprising to see these efficiency measures vary by several fold in either direction if direct measures were feasible. Nevertheless, the magnitude of the energy differences observed (between 10x and 100x) would show a considerable NMC advantage even if these estimates were off considerably.

3. CONNECTING RANDOM WALKS AND PIDES

While RW solutions to PIDEs have mixed appeal to conventional computing programmers, they have been utilized to provide solutions in a variety of fields, including computer science, physics, medicine, and operations research [21].[1] The decision between using a deterministic approach and a random walk approach is a complicated and important question. However, this question is beyond the scope of this paper. Rather, we aim to demonstrate that NMC can efficiently implement random walks and, consequently, are able to solve a variety of PIDEs.

Before showcasing the probabilistic solution to various PIDEs and tying these to our NMC RW algorithm, we motivate the relationship with an example from physics. RW methods take their inspiration from the fundamental relationship between Brownian motion and the heat equation. Consider the one-dimensional heat equation with initial condition given by $f(x)$:

$$\frac{\partial}{\partial t}u = \frac{1}{2}\frac{\partial^2}{\partial x^2}u, x \in R$$

$$u(0,x) = f(x).$$

Equation 1

Let $W(t)$ be a standard Brownian motion on R. The key relationship relates an expectation (i.e. expected or average value) involving $W(t)$ with the solution u:

$$E[f(W(t))|W(0) = x] = \frac{1}{\sqrt{2\pi t}}\int f(y)\exp\left(-\frac{(y-x)^2}{2t}\right)dy = u(t,x).$$

Equation 2

This probabilistic representation allows us to approximate the function $u(t,x)$ using RWs. To do so, we construct a DTMC $X(j\Delta t)$ that approximates the process $W(t)$. For each spatial location x_i, several RWs starting at x_i are generated from the Markov chain. Letting $X_{m,i}$ represent the m^{th} RW generated starting from location x_i, the Monte Carlo approximation gives

$$u(j\Delta t,x_i) = E[f(W(j\Delta t))|W(0) = x_i] \approx \frac{1}{M}\sum_{m=1}^{M} f(X_{m,i}(j\Delta t)).$$

Equation 3

The traditional Monte Carlo approximation would need only discretize time and sample paths of the process $W(t)$ over a continuous space [14]. As we will discuss further on, a finite discretized space is needed for compatibility with neuromorphic hardware.

---

[1] Due to their broad relevance, terminology such as "Monte Carlo", "random walks", and other terms may have specific meanings in some fields, so to give clarity to the methods that follow, we emphasize that we are employing discrete-time, finite state space Markov chain approximations to stochastic processes underlying particular PIDEs. These Markov chains are used to generate several random walks. These random walks are evaluated in a Monte Carlo fashion to estimate an expectation.

Regardless of modifications needed for NMC implementation, this simple result can be extended to a more computationally challenging set of problems. Consider the family of PIDEs defined by the equation

$$\frac{\partial}{\partial t}u(t,x) = \frac{1}{2}\sum_{i,j}\left(aa^{\top}\right)_{i,j}(t,x)\frac{\partial^2}{\partial x_i \partial x_j}u(t,x) + \sum_i b_i(t,x)\frac{\partial}{\partial x_i}u(t,x) + \lambda(t,x)\int \left(u(t,x+h(t,x,q)) - u(t,x)\right)\phi_Q(q;t,x)dq + c(t,x)u(t,x) + f(t,x),\ x \in R^d, t \in [0,\infty).$$

Equation 4

As with Equation 1, there is an underlying stochastic process, albeit slightly more complicated than just Brownian motion. The stochastic process related to this PIDE is

$$dX(t) = b(t,X(t))dt + a(t,X(t))dW(t) + h(t,X(t),q)dP(t;Q,X(t)).$$

Equation 5

The process $X(t)$ is defined by a drift, diffusion, and a non-local jump. In this form, b gives the drift and a gives the diffusion. The process $W(t)$ is a Brownian motion with respect to the underlying space, in this case R^d. The term $P(t;Q,X(t))$ is a Poisson process with parameter given by $-\int_0^t \lambda(s,X(s))ds$ and the function h describes the non-local jump awarded whenever the Poisson process fires. This stochastic process is readily visualized in **Figs. 2a-c** for constant values of $b,a,$ and h. The jump value h need not be constant and can even be random as seen in **Fig. 2d** (Q can be interpreted as a random variable corresponding to the random jump mark amplitude of a compound Poisson process). The final two panels showcase when the jump value is drawn uniformly over $\{-3,-2,\ldots,2,3\}$. We note that while c does not appear in Equation 4, it can often be interpreted as an absorption or killing term, demonstrated in **Fig 2e**. A discussion on this interpretation can be found in **Supplementary Note 2**.

Pairing Equation 4 with the initial condition $u(0,x) = g(x)$, under certain conditions the solution to the initial value problem can be represented as

$$u(t,x) = E\left[g(X(t))\exp\left(\int_0^t c(s,X(s))ds\right) + \int_0^t f(s,X(s))\exp\left(\int_0^s c(l,X(l))dl\right)ds \,\middle|\, X(0) = x\right].$$

Equation 6

A proof for the one-dimensional case can be found in **[SN2]**.

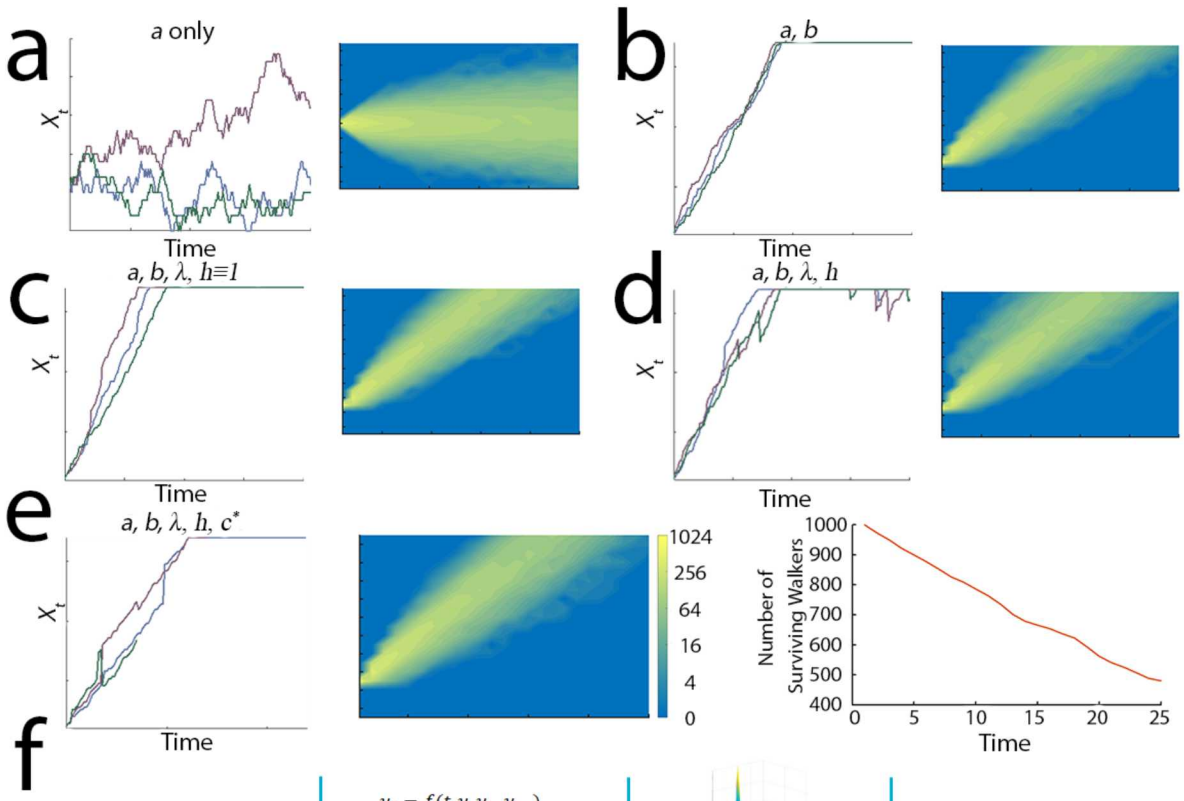
Various special cases of this result exist. A particular interesting special case arises when considering the steady-state version of Equation 4, where $\frac{\partial}{\partial t}u = 0$ and t does not appear as an argument in all functions. Setting $c = 0$ and considering this case as a boundary-value problem with $u(x) = v(x)$ on the boundary of some domain D, the solution can be shown to take the form

$$u(x) = E\left[v\left(X(T_x)\right) + \int_0^{T_x} f(X(s))ds \middle| X(0) = x\right].$$

Equation 7

Here, $X(t)$ is the process given by Equation 5 with t omitted as the first argument in a, b, and h. Since time is still an argument for the process, the probabilistic solution requires the use of the stopping time T_x, or the time for which the random process $X(t)$, starting at $X(0) = x$ exits the domain D. A proof for the one-dimensional case can be found in **[SN2]**.

These PIDEs are important within many application domains, including particle physics, quantitative finance, molecular dynamics, among others. When viewed probabilistically, the steady-state problems are particularly interesting for neuromorphic, as running RWs to the long times required for steady-state solutions are often computationally prohibitive on conventional hardware.



f

Problem	Approximation	Visualization	Error/Convergence	
PDE Ground Truth	$u_t = f(t,u,u_x,u_{xx})$ $u(t,x) = \mathbb{E}[g(t,X_t)\|X_0 = x]$			
Sampling stochastic process	$u(t,x) \approx \frac{1}{M}\sum_{i=1}^{M} g(t,X_t^i);\ X_0^{(i)} = x$		$\frac{1}{\sqrt{M}}$	Random Walk Approximations
Time discretization of RWs	$u(j\Delta t,x) \approx \frac{1}{M}\sum_{i=1}^{M} g(j\Delta t,X_{j\Delta t}^i);\ X_0^{(i)} = x$		$\sqrt{\Delta t}$	
Spatial discretization of RWs	$u(j\Delta t,x_k) \approx \frac{1}{M}\sum_{i=1}^{M} g(j\Delta t,\hat{X}_{j\Delta t}^i);\ \hat{X}_0^{(i)} = x_k$		$\frac{1}{2}j\Delta t\Delta s$	Neuromorphic Specific
Finite range of RWs	$u(j\Delta t,x_k) \approx \frac{1}{M}\sum_{i=1}^{M} g(j\Delta t,\hat{X}_{j\Delta t}^i);\ \hat{X}_0^{(i)} = x_k$		varies	
Limited precision probabilities	$\mathbb{P} \propto \frac{1}{256}$		varies	Platform Specific

Figure 9: Random walk processes are well-suited for NMC, and the inclusion of different terms in the stochastic process yields random walks with differing behavior. For (a)-(e), left panel shows illustrative random walks for 100 timesteps; right panel shows density of 1000 walkers run on Loihi for 25 timesteps. (a) Including only an a term yields basic diffusion; (b) Including a and b yields diffusion with drift. (c-d) The inclusion of λ and h allows the random walk to 'jump' for discontinuous movements. (e) The c term under some conditions can yield walker removal. (f) Sources of discretization in all stochastic processes (of either conventional or neuromorphic sources) impacts the accuracy and convergence of expectation solution for the PDE. The first row details the Monte Carlo order of convergence; the second row is the order of convergence for the Euler-Maruyama discretization method; the third row is a best-case scenario estimate for error accrued due to discretizing space; the fourth and final rows merely indicate that some problems could have additional error due to enforcing a finite state space or due to reduced precision on neuromorphic platforms. For further discussion, see Methods.

Non-Zero Terms in Equation 4	Example Application
Time-dependent problems	
a,b,c,f	Stock Option Pricing **[6]**
λ,b,c,f,h	Boltzmann Flux Density **[SN3]** Reduced Problem, **Fig. 3a-d**.
a, c	Heat Equation with Dissipation (See **Fig. 4c**)
Steady-state problems	
a,f	Electrostatic Scalar Potential, Heat Transport, or Simple Beam Bending [34]
λ, b,c,f,h	Particle Fluence **[SN3]** Reduced problem, **Fig. 3e-i**.

Table 6: Examples of applications involving a PIDE in the form of Equation 4. This table is not exaustive and includes only a sample of possible applications. In this paper, we utilize a random walk method to solve two heat transport problems and a reduced problem for both the Boltzmann particle angular flux density problem and the angular fluence problem.

A DTMC approximating Equation 5 is compatible with the neural algorithm we described for diffusion (**Fig 1d**). In particular, the drift (b) and non-local diffusion terms (λ,h) can naturally be reflected within the definition of the mesh and transition probabilities (**Fig 1c**), in effect providing those extensions to diffusion. Similarly, non-conservation of walkers (walker absorption or creation) can be easily integrated into the system we described. Such a situation may be desirable when the form of c lends itself towards an absorption interpretation.

To approximate Equation 5 with a DTMC, one must employ some sort of temporal and spatial discretization scheme. Having NMC approximate the DTMC introduces additional sources of uncertainty (**Fig 2f**). Specifically, the finite node structure of NMC architectures forces the DTMC to have a finite state space. In one dimension, this equates to having a maximum and minimum value in the state space. The error of enforcing a finite state space for the DTMC would vary from

application to application. The discrete state space arising from the DTMC also introduces error depending on the problem at hand. If the state space of the random walk is already discrete, it introduces no error. In the continuous case, it could introduce error on the order of $\frac{1}{2}\Delta t \Delta s$ on each time step in the best-case scenario (see **Methods**). Additional error could arise from hardware specific limitations. For instance, the IBM's TrueNorth and Intel's Loihi pseudo-random number generators that we use are effectively limited to 8 bits.

Both conventional simulations, which model each random walker independently and track the evolution of state variables, and our neuromorphic simulations, which model the parallel evolution of random walkers over a state-space represented by the neural circuit, are impacted by each of these error sources. However, the high numerical precision of conventional processing minimizes the impacts of discretizing the values and ranges of state variables, making the dominant errors due to time discretization and the number of random walkers. In contrast, our neuromorphic implementation enables a very large number of walkers at negligible cost, but the dedication of neurons to explicitly representing state variables raises the cost of reducing the meshing error. The implication of these errors will differ considerably across applications in practice.

3.1. Mathematical Methods

Here, we discuss a general approach to creating a discrete-time finite state space Markov chain approximation to a jump-diffusion SDE.

The success of approximating solutions to PIDEs hinges on the ability to implement a random walk approximation of the underlying stochastic process for each PIDE. While there are many different ways to construct such an approximation, we will discuss the basics and point out where variations can occur.

3.1.1. Construction

We will explain a basic method for constructing a discrete-time finite state space Markov chain approximation to the following one-dimensional SDE:

$dX(t) = b(t,X(t))dt + a(t,X(t))dW(t) + h(t,X(t),q)dP(t;Q,X(t),t)$.

For discussion, assume that the domain of interest is R. It is important to note that this means the process $X(t)$ will take on values in the whole of R. The Markov chain approximation we construct will take values on a finite set. This means we will need to divide the real line into a finite number of intervals to represent our state space and, using [eq:1d_sde], determine the probability of transition between these intervals.

However, the determination of this probability is subtle – a representative location for the interval is needed, and neuromorphic hardware constraints may limit the number of allowable transitions. Given a starting location, there is a non-zero probability that the process $X(t)$ can transition to any interval on the real line. If the number of allowable transitions in the Markov chain is limited, care must be taken to conserve probability. That is, the particle must transition to somewhere and the probability of the allowable transitions must sum to 1.

We will carefully explore the nuances of the approximation, first exploring a countable state space and then restricting to a finite one. Specifically, we will follow this order:

1. Define a countable state space for the Markov chain;
2. Determine neighbors for each state in the Markov chain;
3. Calculate the probability for each transition in the chain; and
4. Restrict to a finite state space.

The ultimate artifact of construction is a transition matrix among the states of a Markov chain.

3.1.2. Countable state space for the Markov chain.

To begin our approximation for [eq:1d_sde] we must chop the real line into a sequence of "bins" or "nodes." We note that this discretization does not have to be uniform. In the interest of keeping this discussion simple, we elect to make uniform divisions. Suppose we wish to make uniform divisions of the real line, starting at 0 of a selected size Δx. From zero, in both directions, we count out the edges of our bins by increments of Δx. Since the probability calculations will require a starting point, we need to choose a location to represent each of these intervals. We use the midpoints of these intervals to represent the countable state space. In Figure 10, these states are illustrated by circles.

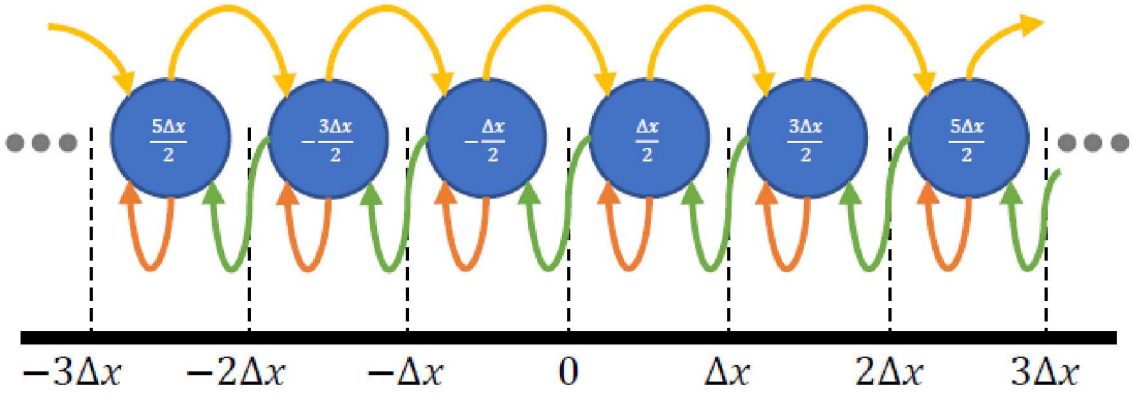


Figure 10: Illustration on the creation of a Markov Chain on the Real Line

3.1.3. Neighbors for each state in the Markov chain.

We must now identify the neighbors of each state in our space. As previously discussed, given a starting location, the process $X(t)$ has a non-zero probability of transitioning to any of our defined intervals on the real line. If we were to mimic this, then each state in our state space would be a neighbor to all other states. That is, the graph representing the possible transitions between state spaces would be a complete graph.

However, neuromorphic hardware and practical limitations (i.e., fan-in/fan-out considerations) suggest that we cannot allow each state space to transition to all others. This means a choice must be made. In the context of neuromorphic hardware, we must decide how much fan-out we want. In the examples in the text, we typically allowed transitions between adjacent states and back to the original state. For this discussion, we allow transitions to the left and right, and back to itself (see arrows in Fig. [fig:1dline]).

3.1.4. Calculation of transition probabilities.

Our first two steps in the calculation of the Markov chain approximation involved two independent choices: a choice of Δx and a choice of neighboring states. To calculate the probabilities, a dependent choice will be made based on the previous selections. Namely, we must select an appropriate time discretization size Δt for the discrete time Markov chain.

Again, recalling that [eq:1d_sde] can transition to any interval on the real line given a starting position, we must choose Δt small enough so that the probability of transitioning to intervals outside of the defined neighbors is smaller than some threshold probability. For all of our examples in the text, we choose Δt small enough so that transition outside of the chosen neighbors is less than 0.05.

However, the choice of Δt may also depend on another factor. If there is a Poisson process, that is if $dP(t;Q,X(t),t)$ appears in the SDE, then we also want to choose Δt small enough so that we can be reasonably sure that *at most* one Poisson event can occur in any time window. Again, we ensure Δt is selected so that the probability of more than one Poisson event occurring in any time window is less than 0.05.

To that end suppose that x_i and x_j are nodes and that x_j is a neighbor of x_i. What is the probability that x_i transitions to x_j? For simplicity, we will assume that h is deterministic. Then, using [eq:1d_sde], the probability of x_i transitioning to x_j in the time interval $[t,t+\Delta t]$ is the sum of two probabilities. Given $X(t)=x_i$, the first is the probability that $X(t+\Delta t)$ is in the interval represented by x_j and that no Poisson jump occurs. The second is the probability that $X(t+\Delta t)$ is in the interval represented by x_j and some Poisson jump occurred.

To help calculate these probabilities, we will appeal to the Euler-Maruyama simulation method for SDEs. This is a discretization method for simulating sample paths of SDEs. The method works similar to the Euler method in that the value of $X(t)$ is assumed constant over some interval $[t,t+\Delta t]$ and an increment is calculated from this assumption. For [eq:1d_sde], given $X(t)=x_i$ and assuming that h is deterministic, the Euler-Maruyama method gives

$$X(t+\Delta t) \approx x_i + b(t,x_i)\,\Delta t + a(t,x_i)\,W(\Delta t) + h(t,x_i)\,\mathbb{1}_{P(t;Q,x_i,t)},$$

where $\mathbb{1}_{P(t;Q,x_i,t)}$ represents the indicator function of whether the Poisson process P fired in the time window $[t,t+\Delta t]$ given that $X(t)=x_i$, and $W(\Delta t)$ is a normal random variable with mean 0 and variance Δt. Put another way, if the Poisson process *does not fire*, then $X(t+\Delta t)$ is assumed to be a normal random variable with mean $x_i + b(t,x_i)\Delta t$ and variance $a^2(t,x_i)\Delta t$. If the Poisson process *does fire*, then $X(t+\Delta t)$ is assumed to be a normal random variable with mean $x_i + b(t,x_i)\Delta t + h(t,x_i)$ and variance $a^2(t,x_i)\Delta t$.

Phrased in this way, calculation of the appropriate probabilities is clearer. Let

$$p_J(x_i,t,\Delta t) = \left(\int_t^{t+\Delta t} \lambda(s,x_i)\, ds \right) exp\left(- \int_t^{t+\Delta t} \lambda(s,x_i)\, ds \right).$$

This is the probability of a single Poisson event occurring in the window $[t,t+\Delta t]$ given that $X(t) = x_i$ is constant on the time interval. Denote $N(\mu,\sigma^2)$ denote a normal random variable with mean μ and variance σ^2. Then,

$$p_{ij}(t) := P \quad [x_i \to x_j \mid t \to t+\Delta t] \approx \left(1 - p_J(x_i,t,\Delta t)\right) P\left[N\left(x_i + b(t,x_i)\Delta t,\ a^2(t,x_i)\Delta t\right)\right]$$
$$XXXXXXX + p_J(x_i,t,\Delta t) P\left[N\left(x_i + b(t,x_i)\Delta t + h(t,x_i),\ a^2(t,x_i)\Delta t\right)\right].$$

This is a probabilistic representation of the intuition we discussed: the probability of transitioning is the sum of two probabilities, one of ending up in the interval with no jump and one of ending up in the interval having experienced a jump.

In the the event that the jump h is random, a similar construction can be made taking care to sum or integrate over the distribution of rewards. Our steady-state particle transport problem is an example of this concept.

Letting $t_j = j\Delta t$, [eq:jump_probability] can be used along with the probability of no jumps occurring

$$p_0(x_i,t_j,\Delta t) = exp\left(- \int_{t_j}^{t_j} + \Delta t \lambda(s,x_i)\, ds \right)$$

to ensure that the probability of more than one jump occurring $(1 - p_0 - p_J)$ is less than 0.05 for all x_i and t_j. Similarly, [eq:trans_prob] can be used to ensure the probability of transitioning to neighbors other than the allowed transitions is less than 0.05 for all x_i. The time discretization Δt should be chosen to satisfy both of these constraints.

Using [eq:trans_prob], a transition tensor $C(t_l) = (p_{ij}(t_l))$ can be constructed representing the Markov chain. For each fixed t_l, we must conserve probability – the probability of transitioning from state x_i to some other state must sum to 1. In the Markov chain, this means that $\sum_j p_{ij}(t_l) = 1$. However, since $X(t)$ informed our transition probabilities and $X(t)$ can transition to any interval on the real line, calculating the transition probabilities for just a subset of the possible transitions may mean that these probabilities do not sum to 1. While Δt was chosen so that this probability of transition is small, we must require a sum of 1 to ensure the Markov chain is well defined.

There are various ways to conserve probability at this point. One possibility is to just normalize the rows. Another possibility is to add the missing amount to one of the transitions. Unless stated otherwise in our example discussions, we calculate as follows. If, in our example, we were calculating the transition of probability to the left node, we would calculate the probability of transitioning to the left node *or beyond*. Similarly for the right node. Again, since Δt has been chosen so that

transitioning more than a single space to the left or right is small, the error accrued from this assignment is also small.

It is important to call out that C does depend on the time step t_l whenever a, b, λ, or h depend on t. When these functions do not depend on time, a static transition matrix is created. If time dependence exists and if a maximum desired time is known, the transition tensor C can be collapsed into a single matrix. For example, if there are 10 possible state spaces and it is only desired to simulate for 100 time steps, then the one hundred 10×10 transition matrices would become a single transition matrix over a state space of size 1000.

3.1.5. Restriction to a finite state space.

For some problems, like our heat transport on the sphere or time-dependent particle transport examples, the construction above yields a finite state space. However, situations may arise where the method we have described yields a countably infinite state space. When considering hardware limitations, like the finite number of nodes on a spiking neuromorphic platform, it may be necessary to reduce to a finite state space.

Continuing our discussion of [eq:1d_sde] on the real line, we would need to select a finite subset of our states to transition between. Due to the construction of neighbors, we probably would select a minimum interval and a maximum interval, keeping all states between. Without loss of generality, suppose that the states are ordered and that the minimum state corresponds to x_1 and the maximum state corresponds to x_N.

For each t_l, we pluck out the $N \times N$ section of the transition matrix $C(t_l)$ corresponding to our truncated state space. To conserve probability, for each state we add the total probability of a transition to a state less than x_1 to the probability of transitioning to x_1. Similarly for a transition to a state greater than x_N. In this example where transitions are restricted to the left, right, and to the same state, this addition of probabilities only occurs on the endpoints.

Once the transition matrix is finalized, it can be used to sample some number M of random walks. If x_j is some node in the state space, then we write $X_{j,i}(k\Delta t)$ for the location of the i^{th} random walk that started at position x_j at time $k\Delta t$.

3.2. Utilizing Sampled Random Walks to Approximate PIDE Solution

Consider the probabilistic solution from Theorem [thm:class_1] in Supplementary Note 2:

$$u(t,x) = E\left[g(X(t))exp\left(\int_0^t c(s,X(s))ds\right) | X(0) = x\right] + E\left[\int_0^t f(s,X(s))exp\left(\int_0^s c(l,X(l))dl\right)ds \,|X(0) = x\right].$$

We would like to evaluate $u(t_i,x_j)$ via this expectation using the Monte Carlo method and the sampled random walks for each position x_j in the mesh and time point t_i. Since our random walks occur over discrete time points and occupy discrete locations, we do this via Riemann sum

approximations for the integrals. If there were M total random walks sampled that started on position x_j, then

$$u(t_i,x_j) \approx \frac{1}{M}\sum_{l=1}^{M}\left[g(X_{j,l}(i\Delta t))exp\left(\sum_{k=0}^{i}c(t_k,X_{j,l}(k\Delta t))\Delta t\right) + \sum_{k=0}^{i}f(k\Delta t,X_{j,l}(k\Delta t))exp\left(\sum_{s=0}^{k}c(s\Delta t,X_{j,l}(s\Delta t))\Delta t\right)\Delta t\right].$$

3.3. Brief Commentary on Accuracy of Approximation

Here, we evaluated the implications of using a discrete spatial mesh for approximating random walks that ideally would be continuous valued. Such approximations are implicit in any numerical implementation of random walks on a system with finite precision, although since our implementation directly implements a discrete mesh to describe the state-space, it is necessary to consider the numerical implications. We approximate a jump-diffusion process with with a discrete-time Markov chain. Colloquially called the (jump) diffusion approximation, weak convergence results were found by Skorokhod [33]. Analytic, rather than probabilistic, proofs were reconsidered for the diffusion approximation determining an order of convergence of $O(1/\sqrt{n})$ [18].

Fixing some initial condition and setting $\Delta x = 1/n$, let $X_n(t)$ represent the Markov chain approximation of [eq:1d_sde]. By saying that $X_n(t)$ converges weakly to $X(t)$, we mean that as $n\to\infty$, or rather as $\Delta x\to 0$, the transition density of X_n converges to that of X. If a function Ψ is continuous and bounded and if X_n converges weakly to X, then $E^*[h(X_n)]$ converges weakly to $E[h(X)]$, where E represents the outer expectation (or measure). This, effectively, implies that we have convergence of [eq:ivp_approx] to [eq:ivp_sol_m] as we decrease Δx. Beyond the scope of this work, error bounds for continuous time Markov chain approximations ($\Delta t\to 0$) have been found for PIDEs involving fractional time operators (Kelbert, Konakov, and Menozzi 2016).

Setting aside these concepts of order of convergence, we would like to demonstrate an error estimate on using the discretization of space in the *best case scenario*. To that end, fix some time t and a position x. Suppose we wish to approximate

$$u(t,x) = E[\Psi(X(t)) \mid X_0 = x]$$

by sampling M paths of $X(t)$. Let X_i be the i^{th} sampled path. Then the Monte Carlo approximate solution is

$$u_M(t,x) = \frac{1}{M}\sum_{i=1}^{M}\Psi(X_i(t)).$$

To mimic the discrete spatial approximation in the best case scenario, let's assume that there exist true random paths $\Psi(X_i(t))$ and that our approximation "snaps" or rounds the true value of $\Psi(X_i(t))$ to the nearest point on a spatial grid with size Δx. We will denote this rounded process by $\hat{\Psi}(X_i(t))$ and the resulting approximate Monte Carlo solution by

$$\hat{u}_M(t,x) = \frac{1}{M}\sum_{i=1}^{M} \hat{\Psi}(X_i(t)).$$

In this manner, at time t, the process $\hat{\Psi}(X_i(t))$ is at most $\Delta x/2$ away from the true sample path $\Psi(X_i(t))$. The error of this best case approximation is estimated as

$$\begin{aligned}\left|u(t,x) - \hat{u}_M(t,x)\right| &\leq \left|u(t,x) - u_M(t,x)\right| + \left|u_M(t,x) - \hat{u}_M(t,x)\right| \\ &\leq \left|u(t,x) - u_M(t,x)\right| + \frac{1}{M}\sum_{i=1}^{M}\left|\Psi(X_i(t)) - \hat{\Psi}(X_i(t))\right| \\ &\leq \left|u(t,x) - u_M(t,x)\right| + \frac{\Delta s}{2}.\end{aligned}$$

The first term on the right hand side is the Monte Carlo error and is on the order of $1/\sqrt{M}$. The second term is the error accrued due to forcing our process $\Psi(X(t))$ to snap to a grid. Notably, it cannot be controlled by the number of samples M and can only be made small with a sufficiently small grid.

When using the Markov chain approximation, however, we do not have the actual path and instead calculate the next value in the path by assuming the random walk is in the center of a voxel and then travels to the center of another voxel with a probability determined from landing anywhere in the voxel. The error that arises by this method would increase on each time step. Even in the best case scenario, the error accrued on the j^{th} time step would be dependent on the function Ψ applied to the underlying spatial grid. If Ψ is the identity, and we are snapping a true sampled path $X_i(t)$ to a grid, then the error accrued in an absolute best case scenario for the j^{th} time step would be on the order $j\Delta t\Delta x/2$.

Reducing to a finite state space is problem specific. The error heavily depends on the application and how the finite states are selected. Additionally, the examples we consider do not require this truncation. As such, we will not explore the consequences of a finite state space here.

3.4. Supplementary Note 2: Connecting a Class of PIDEs with a Probabilistic Representation

The underlying tools used here are the stochastic analogs of methods from differential calculus. Specifically, the main tool leveraged is Itō's rule (sometimes called Itō's lemma or Itō's formula). This landmark result from stochastic calculus is often referred to as the stochastic chain rule and serves the same purpose as the traditional chain rule in calculus. The two proofs in this note follow the same general road map. Itō's rule is used to determine a form for $du(t,X(t))$, where $X(t)$ is a stochastic process. Once this is found, integration in time followed by an expectation yields the final result.

In the results provided below, we assume that a unique solution to the problem exists. Then, we show that the unique solution has a probabilistic representation. We do not address conditions for the existence and uniqueness of a solution. Classical solutions to these types of problems are

discussed in [13, 35]. For this style of problem with no integral term, conditions for existence, uniqueness, and the probabilistic representation are covered in [14].

The results for probabilistic representations were heavily influenced by techniques used in financial math. As such, the results are typically concerned with final value problems. Indeed, [35] contains results of this flavor. The primary source we use for the results shown is [16]. This text determines the Kolmogorov equations for a generic jump-diffusion process. Using these derivations, the author discusses a probabilistic representation for the final value problem of the PIDE considered in the main text.

It is beyond the scope of this work to provide a full introduction to stochastic calculus. However, we provide a few short details of stochastic differential equations for intuition purposes. Consider the stochastic differential equation

$$dX(t) = b(t,X(t))\,dt + a(t,X(t))dW(t) + h(t,X(t))dP(t).$$

In this equation, $W(t)$ is a Brownian motion and $P(t)$ is a Poisson process. Those with some background in white noise processes may feel uneasy with this equation since Brownian motion is nowhere differentiable. Therefore the concept of "$dW(t)$" does not make much sense. However, this equation is merely shorthand for the integral equation

$$X(t) = X(0) + \int_0^t b(s,X(s))\,ds + \int_0^t a(s,X(s))\,dW(s) + \int_0^t h(s,X(s))\,dP(s).$$

In this context, $dW(t)$ feels more like the concept of Riemann-Stieltjes integration. Indeed, integration against Brownian motion can be shown to make mathematical sense. Similarly, integration against a Poisson measure is well defined. The integrals with respect to Brownian motion and the Poisson process are called stochastic integrals. The expectation of the Brownian motion one is always zero. When the Poisson measure has mean zero, the expectation of the Poisson integral can also be shown to be zero.

Returning to the SDE, we are now ready to apply some intuition. The term $b(t,X(t))$ is the drift, or velocity, term. This describes the overall trend of the process $X(t)$. The next term, $a(t,X(t))$ is the diffusion term. This term describes the noise associated with the process $X(t)$. The final term $h(t,X(t))$ is a special term. This describes the value of any jumps in the process. The Poisson process $P(t)$ determines the times of the jumps. For further reading, (Wiersema 2008) provides an overview of the stochastic calculus; an in-depth approach covering stochastic integration against more general processes is given in [26].

We have organized this note as follows. In Section 3, we prove a representation theorem for the time-dependent initial value problem. In Section 4, we prove an analogous representation theorem for a special case steady-state boundary value problem. We adopt the notation from (Hanson 2007). Hence, in all that follows, allow a semicolon in an argument to denote that the function might retain the variables to the right of the semicolon. An example of this retention occurs in our particle transport problems (see Supplementary Note 2). In these problems, the change in direction of transport after a scattering event depends on the previous direction. Hence, the previous direction appears in the probability density function.

3.4.1. A Probabilistic Solution for an Initial Value Problem PIDE

The following result is discussed for a final value problem in [16]. To obtain the result for the initial value problem considered, we would only need to perform a change of variables. Rather than citing the result, we instead prove the result for the initial value problem with a slight alteration. Namely, we generalize the probability space so that the result can be directly applied to geometries like the sphere.

Let (Σ,F,P) be a probability space and suppose $x \in \Sigma \subseteq R$ and $t \in [0,\infty)$. Let $a,b,\lambda,c,f,u:[[0,\infty) \times \Sigma] \to \Sigma$ and $g:\Sigma \to \Sigma$. Assume that $a \geq 0$. Let $W(t)$ be a Brownian motion with respect to the probability space. Let $P(t;Q,x)$ be a Poisson process such that

$$E[dP(t;Q,x,t)] = \lambda(t,x)dt,$$

where Q is the jump-amplitude mark random variable of the process with domain in Q and probability density function ϕ_Q. Let $h:[[0,\infty) \times \Sigma \times Q] \to \Sigma$. Define the stochastic process

$$dX(t) = b(t,X(t))dt + a(t,X(t))dW(t) + h(t,X(t),q)dP(t;Q,X(t),t).$$

Consider the initial value problem

$$\begin{aligned} \frac{\partial}{\partial t}u(t,x) &= \frac{1}{2}a^2(t,x)\frac{\partial^2}{\partial x^2}u(t,x) + b(t,x)\frac{\partial}{\partial x}u(t,x) \\ &\quad + \lambda(t,x)\int (u(t,x+h(t,x,q)) - u(t,x))\phi_Q(q;t,x)dq \\ &\quad + c(t,x)u(t,x) + f(t,x), \\ u(0,x) &= g(x). \end{aligned}$$

Suppose:

- The functions a, b, and h are continuously differentiable in all arguments and their spatial gradients are bounded.
- The functions c,f,g are bounded and continuous almost everywhere.

Then, if a unique solution to [eq:ivp_class_1] exists, it is given by

$$\begin{aligned} u(t,x) &= E\left[exp\left(\int_0^t c(s,X(s))ds\right)g(X(t)) \mid X(0) = x\right] \\ &\quad + E\left[\int_0^t f(s,X(s))exp\left(\int_0^s c(l,X(l))dl\right)ds \mid X(0) = x\right]. \end{aligned}$$

Theorem S2.1.1. *Let $(\Sigma, \mathcal{F}, \mathbb{P})$ be a probability space and suppose $x \in \Sigma \subseteq \mathbb{R}$ and $t \in [0,\infty)$. Let $a, b, \lambda, c, f, u : [[0,\infty) \times \Sigma] \to \Sigma$ and $g : \Sigma \to \Sigma$. Assume that $a \geq 0$. Let $W(t)$ be a Brownian*

motion with respect to the probability space. Let $P(t;Q,x)$ be a Poisson process such that

$$\mathbb{E}\left[\mathrm{d}P(t;Q,x,t)\right] = \lambda(t,x)\mathrm{d}t,$$

where Q is the jump-amplitude mark random variable of the process with domain in $\mathcal{Q}$ and probability density function ϕ_Q. Let $h : [[0,\infty) \times \Sigma \times \mathcal{Q}] \to \Sigma$. Define the stochastic process

$$\mathrm{d}X(t) = b\left(t, X(t)\right)\mathrm{d}t + a\left(t, X(t)\right)\mathrm{d}W(t) + h\left(t, X(t), q\right)\mathrm{d}P\left(t; Q, X(t), t\right). \tag{S2.1}$$

Consider the initial value problem

$$\begin{aligned}\frac{\partial}{\partial t}u\left(t,x\right) &= \frac{1}{2}a^2\left(t,x\right)\frac{\partial^2}{\partial x^2}u\left(t,x\right) + b\left(t,x\right)\frac{\partial}{\partial x}u\left(t,x\right)\\ &\quad + \lambda\left(t,x\right)\int\left(u\left(t,x+h\left(t,x,q\right)\right) - u\left(t,x\right)\right)\phi_Q\left(q;t,x\right)\mathrm{d}q\\ &\quad + c\left(t,x\right)u\left(t,x\right) + f\left(t,x\right),\\ u\left(0,x\right) &= g\left(x\right).\end{aligned} \tag{S2.2}$$

Suppose:

- *The functions a, b, and h are continuously differentiable in all arguments and their spatial gradients are bounded.*
- *The functions c, f, g are bounded and continuous almost everywhere.*

Then, if a unique solution to (S2.2) exists, it is given by

$$\begin{aligned}u\left(t,x\right) &= \mathbb{E}\left[\exp\left(\int_0^t c\left(s,X(s)\right)\mathrm{d}s\right)g\left(X\left(t\right)\right)\middle|X(0)=x\right]\\ &\quad + \mathbb{E}\left[\int_0^t f\left(s,X(s)\right)\exp\left(\int_0^s c\left(\ell,X(\ell)\right)\mathrm{d}\ell\right)\mathrm{d}s\middle|X(0)=x\right].\end{aligned} \tag{S2.3}$$

Proof

Let u be the unique solution to [eq:ivp_class_1]. Let some $t \in (0,\infty)$ be given. Let $s \in [0,t)$ represent the forward time (or time since zero) and $\tau = t - s$ represent the backward time (or time until t).

Then $\hat{u}(t-s,x) = u(s,x)$ satisfies the final value problem

$$\begin{aligned}-\frac{\partial}{\partial\tau}\hat{u}(\tau,x) &= \frac{1}{2}a^2(\tau,x)\frac{\partial^2}{\partial x^2}\hat{u}(\tau,x) + b(\tau,x)\frac{\partial}{\partial x}\hat{u}(\tau,x)\\ &\quad + \lambda(\tau,x)\int\left(\hat{u}(\tau,x+h(\tau,x,q)) - \hat{u}(\tau,x)\right)\phi_Q(q;\tau,x)dq\\ &\quad + c(\tau,x)\hat{u}(\tau,x) + f(\tau,x),\\ \hat{u}(t,x) &= u(0,x) = g(x).\end{aligned}$$

To simplify notation, denote

$$\Delta\hat{h}(\tau,x,q) = \hat{u}(\tau,x+h(\tau,x,q)) - \hat{u}(\tau,x).$$

For $\rho \in (\tau,t)$, consider the function

$$w(\tau,\rho,X(\rho)) = u(\rho,X(\rho))v(\tau,\rho),$$

where

$$v(\tau,\rho) = exp\left(\int_{\tau}^{\rho} c(l,X(l))\,dl\right).$$

Itō's rule combined with the stochastic product rule yield

$$\begin{aligned} dw(\tau,\rho,X(\rho)) \quad = v(\tau,\rho)\Bigg[\Bigg(&\frac{\partial \hat{u}}{\partial \rho}(\rho,X(\rho)) + b(\rho,X(\rho))\frac{\partial \hat{u}}{\partial x}(\rho,X(\rho)) \\ &\frac{\partial}{\partial x}xxx + \frac{1}{2}a^2(\rho,X(\rho))\frac{\partial^2 \hat{u}}{\partial x^2}(\rho,X(\rho)) \\ &\frac{\partial}{\partial x}xxx + c(\rho,X(\rho))u(\rho,X(\rho))\Bigg)d\rho \\ &xx + b(\rho,X(\rho))\frac{\partial \hat{u}}{\partial x}(\rho,X(\rho))dW(\rho) \\ &xx + \int_{Q} \hat{u}^{\Delta h}(\rho,X(\rho),q)P(d\rho,dq;X(\rho),\rho)\Bigg], \end{aligned}$$

where $P(d\rho,dq;X(\rho),t\rho)$ is the Poisson random measure for the Poisson process; i.e.

$$\int_{Q} P(d\rho,dq;X(\rho),\rho) \equiv dP(\rho;Q,X(\rho),\rho).$$

We rewrite the Poisson random measure as the sum of a mean-zero Poisson random measure and the mean:

$$P(d\rho,dq;X(\rho),\rho) = \hat{P}(d\rho,dq;X(\rho),\rho) + \lambda(\rho,X(\rho))\phi_Q(q;X(\rho),\rho)dqd\rho.$$

Using this in [eq:ito_class_1] and appealing to [eq:backward] yields

$$\begin{aligned} dw(\tau,\rho,X(\rho)) \quad = v(\tau,\rho)\Bigg[&-f(\rho,X(\rho))d\rho + b(\rho,X(\rho))\frac{\partial \hat{u}}{\partial x}(\rho,X(\rho))dW(\rho) \\ &x + \int_{Q} \hat{u}^{\Delta h}(\rho,X(\rho),q)\hat{P}(d\rho,dq;X(\rho),\rho)\Bigg] \end{aligned}$$

We now integrate both sides of this equation for $\rho \in (\tau,t)$ and then take an expectation. Since one is against Brownian motion and the other against a mean-zero Poisson measure, the expectation of the stochastic integrals are zero, yielding

$$E[w(\tau,t,X(t)) - w(\tau,\tau,X(\tau)) \mid X(\tau) = x] = E\left[- \int_{\tau}^{t} v(\tau,\rho) f(\rho,X(\rho))\, d\rho \mid X(\tau) = x \right].$$

The left-hand side of this equation becomes

$$\begin{aligned} E[w(\tau,t,X(t)) - w(\tau,\tau,X(\tau)) \mid X(\tau) = x] \quad &= E\left[\hat{u}(t,X(t)) exp\left(\int_{\tau}^{t} c(l,X(l))dl \right) \mid X(\tau) = x \right] \\ &= -\hat{u}(\tau,x). \end{aligned}$$

Hence, we may rearrange and use the final condition to write

$$\begin{aligned} \hat{u}(\tau,x) \quad &= E\left[\hat{u}(t,X(t)) exp\left(\int_{\tau}^{t} c(l,X(l))dl \right) \mid X(\tau) = x \right] \\ &+ E\left[\int_{\tau}^{t} exp\left(\int_{\tau}^{l} c(\rho,X(\rho))d\rho \right) f(l,X(l))dl \mid X(\tau) = x \right] \\ &= E\left[g(X(t)) exp\left(\int_{\tau}^{t} c(l,X(l))dl \right) \mid X(\tau) = x \right] \\ &+ E\left[\int_{\tau}^{t} exp\left(\int_{\tau}^{l} c(\rho,X(\rho))d\rho \right) f(l,X(l))dl \mid X(\tau) = x \right] \end{aligned}$$

Finally, we set $\tau = 0$ to yield [eq:ivp_sol].

In the main text, we mention that the function c can be interpreted as an absorption term. We provide a little intuition for this declaration here. Consider [eq:ivp_class_1] with λ,h,f all zero and assume $c(t,x) = -k$ for some $k \in R^{+}$. Then, [eq:sde_class_1] becomes

$$dX(t) = b(t,X(t))dt + a(t,X(t))dW(t),$$

and [eq:ivp_sol] becomes

$$u(t,x) = E[e^{-kt} g(X(t)) \mid X(0) = x].$$

Imagine that we randomly kill the process [eq:kill_sde] according to a Poisson process with parameter kt. Then, the survival probability, or the probability that $X(t)$ has not been killed by time t is the probability that no events have occurred by time t, or e^{-kt}. Then, we read [eq:kill_sol] as the average of $g(X(t))$ weighted by the probability that $X(t)$ has not died by time t. Rather than doing the weighting in the averaging, we could push this probability into the process $X(t)$ itself. We do this by terminating the process $X(t)$ according to the Poisson process described.

We finally note that Theorem [thm:class_1] can be extended to multiple dimensions, but it requires significantly more bookkeeping. The extension to higher dimensions allows for the inclusion of multiple jump processes and allows for some correlation between stochastic processes. For examples on higher dimensions and further reading on these Feynman-Kac style results, see [9, 14, 16].

3.4.2. A Probabilistic Solution for a Boundary Value Problem Steady-State PIDE

Here we prove a probabilistic representation theorem for a special case of the previous theorem. This ordinary integro-differential equation is a steady-state boundary value problem where $c = 0$. Before proving the following theorem, we will discuss how this result intuitively follows the previous one.

Theorem 3.4.2: *Let* (Σ,F,P) *be a probability space and suppose* $x \in D \subseteq \Sigma \subseteq R$. *Let* $a,b,\lambda,c,f,g,u:D\to D$. *Assume that* $a \geq 0$. *Let* $W(t)$ *be a Brownian motion with respect to the probability space. Let* $P(t;Q,x)$ *be a Poisson process such that*

$$E[dP(t;Q,x,t)] = \lambda(x)dt,$$

where Q *is the jump-amplitude mark random variable of the process with domain in* Q *and probability density function* ϕ_Q. *Let* $h:[D \times Q]\to D$. *Define the stochastic process*

$$dX(t) = b(X(t))dt + a(X(t))dW(t) + h(X(t),q)dP(t;Q,X(t),t),$$

and the associated stopping time

$$T_x = inf\{t > 0 \mid X(t) \notin D, X(0) = x\}.$$

Consider the boundary value problem

$$\begin{aligned} 0 &= \frac{1}{2}a^2(x)\frac{d^2}{dx^2}u(x) + b(x)\frac{d}{dx}u(x) \\ &\quad + \lambda(x)\int (u(x + h(x,q)) - u(x))\phi_Q(q;t,x)dq \\ &\quad + f(x), \\ u(x) &= g(x), \qquad x \in \partial D. \end{aligned}$$

Suppose:

- *The functions* a, b, *and* h *are continuously differentiable in all arguments and their spatial gradients are bounded.*
- *The functions* f,g *are bounded and continuous almost everywhere.*
- *For any* $x \in D$, $E[T_x] < \infty$.

Then, if a unique solution to [eq:bvp_class_2] exists, it is given by

$$u(x) = E\left[g(X(T_x)) + \int_0^{T_x} f(X(s))\,ds \mid X(0) = x\right].$$

[thm:steady]

To discuss how this result might intuitively follow from the first, consider [eq:ivp_class_1]. If we wanted to consider a steady-state version, we could integrate in time, taking t to infinity. This would effectively give us an ordinary integro-differential equation in x for a time integrated version of u. In terms of the SDE, this would mean we would need to sample $X(t)$ for an arbitrarily long time.

If we then move from an equation on the whole plane to an equation on a more restrictive domain, we would need to somehow ensure that our process $X(t)$ does not travel outside of the domain of interest due to random fluctuations. We could accomplish this by artificially cutting off any sample of the process $X(t)$ when it first exits the domain of interest.

We now see how one might intuitively navigate from the time-dependent result to this one. Now we prove the result to fill in the details.

Proof. Let u be the unique solution to [eq:bvp_class_2] and let $u_{\Delta h}$ be as defined in [eq:delta_h] without the time argument. Let $n \in N$ be given. Denote $T_x \wedge n = min\{T_x, n\}$. By Itō's rule, for $t \leq T_x \wedge n$ we have

$$\begin{aligned} du(X(t)) &= \left(b(X(t))\frac{d}{dx}u(X(t)) + \frac{1}{2}a^2(X(t))\frac{d^2}{dx^2}u(X(t))\right)dt \\ &+ b(X(t))\frac{d}{dx}u(X(t))dW(t) + \int_Q u_{\Delta h}(X(t),q)P(dt,dq;X(t),t). \end{aligned}$$

Similar to the previous problem, we rewrite the Poisson random measure as the sum of a mean-zero Poisson random measure $\hat{P}$ and the mean. We then integrate in time from 0 to $T_x \wedge n$, use [eq:bvp_class_2], and rearrange terms:

$$\begin{aligned} u(X(0)) &= u\big(X(T_x \wedge n)\big) + \int_0^{T_x \wedge n} f(X(s))\,ds \\ &- \int_0^{T_x \wedge n} b(X(s))\frac{d}{dx}u(X(s))dW(s) \\ &- \int_0^{T_x \wedge n}\int_Q u_{\Delta h}(X(s),q)\hat{P}(ds,dq;X(s),s). \end{aligned}$$

The expectation of the stochastic integrals are zero. Hence,

$$\begin{aligned} \lim_{n\to\infty} E[u(X(0)) \mid X(0) = x] &= \lim_{n\to\infty} E\left[u\big(X(T_x \wedge n)\big) + \int_0^{T_x \wedge n} f(X(s))\,ds \mid X(0) = x\right] \\ u(x) &= E\left[\lim_{n\to\infty} u\big(X(T_x \wedge n)\big) + \int_0^{T_x \wedge n} f(X(s))\,ds \mid X(0) = x\right] \\ u(x) &= E\left[u\big(X(T_x)\big) + \int_0^{T_x} f(X(s))\,ds \mid X(0) = x\right] \\ u(x) &= E\left[g\big(X(T_x)\big) + \int_0^{T_x} f(X(s))\,ds \mid X(0) = x\right]. \end{aligned}$$

Ergo [eq:class_2_sol] is justified. The limit and expectation can be interchanged as the argument of the expectation forms a uniformly integrable set (see (Grigoriu 2013)).

Similar to the previous theorem, this can also be extended to multiple dimensions with more bookkeeping. In multiple dimensions, this ordinary integro-differential equation would become a PIDE. See [14] for additional multi-dimensional special cases. The given reference considers problems without an integral term and provides assumptions that ensure the existence and uniqueness of the solution as well as the probabilistic representation. The proof provided here demonstrates that the integral term is not difficult to add in when the unique solution is assumed.

4. RESULTS/EXAMPLES

To demonstrate the ability of neuromorphic hardware to implement the DTMCs required for solving these PIDEs, we provide a handful of examples. These are grouped into two main categories: particle equations and geometries. The results of our simulations on hardware and spiking neuron simulators can be found in Figure 11 and Figure 12. We cover the more salient points of these examples in the next two subsections and relegate the remaining details to **[Supplemental Note 3]**.

4.1.1. Neuromorphic hardware can simulate particle transport

First, we showcase two examples of particle transport equations with probabilistic representations suitable for our spiking algorithm. The first is an initial-value time-dependent problem detailing the angular flux density of a hypothetical particle (Figure 11**a**). Consider a fictional particle that has a property called 'direction'. This direction property takes on the value $+1$ or -1. According to a Poisson process with rate σ_s, the particle can experience a 'scattering' event. When a scattering event occurs, the particle chooses a new direction with uniform probability. A second Poisson process with rate σ_a controls when the particle is absorbed and ceases to exist. These rates correspond to λ and c, respectively, in Equation 4. The function h is represented by the change in direction the particle experiences after a scattering event. Coupled with an initial condition g, a population of these particles obeys the Boltzmann equation for angular flux density (see **[SN3]**).

The angular flux density, $\Phi(t,\Omega)$ is a function of both time t and direction Ω. We will leave the PIDE in **[SN3]**, but it takes the form of **Eq. 3** with $a,b,$ and f all equal to zero. Assigning some initial condition g, the solution is given by

$$\Phi(t,\Omega) = E\left[e^{-\sigma_a t} g(Y(t)) \middle| Y(0) = \Omega\right],$$
$$dY(t) = \omega_{Y(t)} dP(t).$$

Equation 8

The SDE almost behaves like our hypothetical particle. The 'direction' at time t is given by $Y(t)$. $P(t)$ is a Poisson process with parameter $\sigma_s t$, and $\omega_{Y(t)}$ is the random change in direction of the particle after a scattering event given its current direction. The direction remains the same until the Poisson process fires (signaling a scattering event). Once this occurs, the value of $Y(t)$ increments by the random change in direction $\omega_{Y(t)}$. Notably, the random process differs from the hypothetical particle in that it does not account for absorption. Instead, absorption is resolved through the exponential term in the expectation.

We deployed a neural circuit of a DTMC approximating the dynamics of the stochastic process $Y(t)$ on TrueNorth. The description of the random walk and the parameter values used can be found in **[SN3]**. In this scenario, an analytic solution exists. Figure 11**c** shows that the true solution is well approximated by sampling just 1000 random walks per each starting condition. Moving to 10,000 RWs per starting position (Figure 11**d**), we see notable improvement in approximation.

This simplified example of particle transport has broad implications. Directly, if we can well-approximate the analytic solution for this reduced particle transport problem, then it will be possible

to approximate more complicated particle transport problems where no solution is available. To that end, we have examined a second particle transport inspired example for which no analytical solution is readily available.

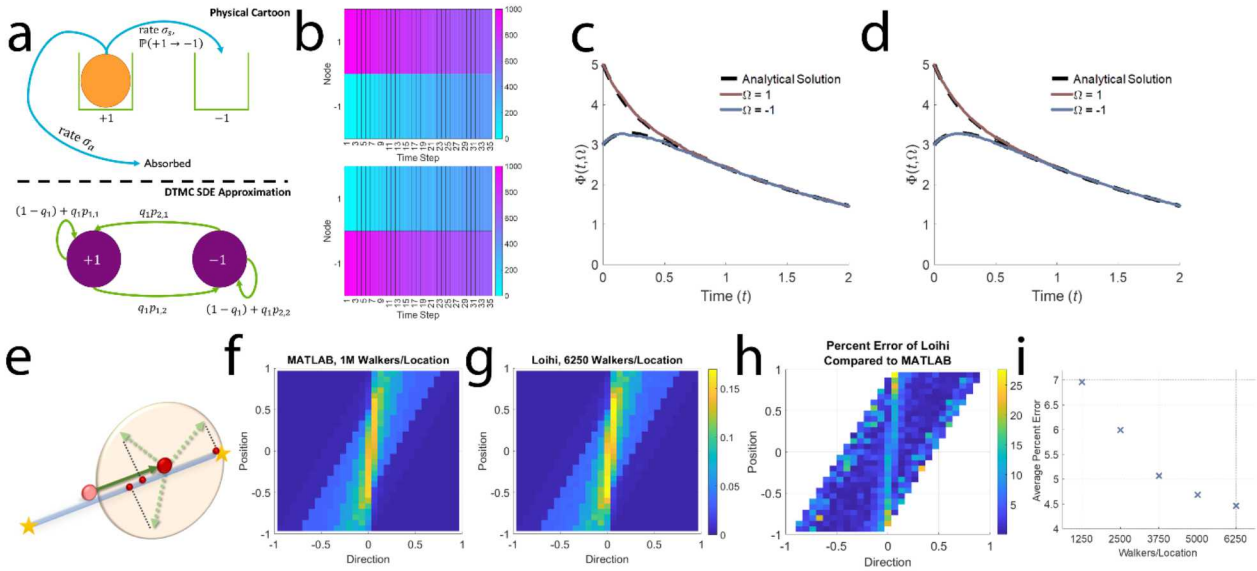


Figure 11: Monte Carlo particle transport simulations on neuromorphic hardware. (a) Non-spatial Boltzmann transition/absorption model (top). Corresponding DTMC approximation for underlying SDE (bottom). (b) Evolution of particles through Boltzmann transitions on TrueNorth. (c) PDE solution calculated through TrueNorth spike data starting 1000 random walkers on each direction. (d) PDE solution calculated through TrueNorth spike data starting 10000 random walkers on each direction. (e) Spatial particle transport model. Particles travel at fixed speed in measured dimension, but at each time step scatter at a random angle, changing their relative velocity. (f) MATLAB approximate solution from DTMC implementation of spatial particle model, 1 million walkers at each starting location (g) Intel Loihi approximate solution from DTMC implementation of spatial particle model, 6250 walkers per starting location (h) absolute error between Loihi and numerical simulation, (i) average percent error between Loihi and numerical simulation as a function of increasing random walkers per starting location.

In our second example, we consider a similar particle. This hypothetical particle is subject to scattering events according to a Poisson process with rate σ_s, however the direction can assume any value in $[-1,1]$ with a uniform distribution. We assume that this particle is not subject to absorption. In addition to direction, this hypothetical particle also has a spatial coordinate. The particle travels at a speed v in the direction Ω updating its position (Figure 11**e**). We seek to find the angular fluence Ψ, or time-integrated flux, in the spatial domain $[-1,1]$ subject to the source term $S(x,\Omega)$.

In terms of Equation 4 (and detailed in **[SN3]**), $\lambda = \sigma_s$, $f = S(x,\Omega)$, $b = v\Omega$, and h is the change in direction after a scattering event given the current direction of travel. The remaining terms, a and c, are zero for this example.

Enforcing absorbing conditions on the boundaries, the angular fluence $\Psi(x,\Omega)$ is a function of position x and direction Ω, and obeys the PIDE in **[SN3]**. The solution may be represented as

$$\Psi(x,\Omega) = E\left[\int_0^{T_x} S(X(u), Y(u))du \middle| X(0) = x, Y(0) = \Omega\right],$$

$$dX(t) = -vY(t)dt,$$

$$dY(t) = \omega_{Y(t)}dP(t),$$

$$T_x = \inf\{t > 0 | X(t) \notin [-1,1], X(0) = x\}.$$

Equation 9

Both $P(t)$ and $\omega_{Y(t)}$ are the same as in the previous example. The SDE in this case describes a process with a position given by $X(t)$ and direction given by $Y(t)$. The position updates with velocity $-vY(t)$. The direction only increases by $\omega_{Y(t)}$ whenever the Poisson process $P(t)$ fires.

We deployed a RW approximation from a DTMC of this joint process on Intel's Loihi platform. Details on the DTMC and parameters used are in **[SN3]**. We completed a 1M walker/location simulation in MATLAB to use as a baseline comparison. One interpretation of the angular fluence is the cumulative density of particles traveling from the source location. From the MATLAB simulation, we see that these particles appear to have mostly traveled with speed v in their original direction assigned by the source, with lessening bands of deviations due to scattering events (Figure 11**f**). Similar to the Boltzmann example on TrueNorth, implementing this simulation on Intel Loihi was able to replicate the numerical examples (Figure 11**g**) with a low overall error (Figure 11**h-i**). This low error in the neuromorphic implementation is of particular importance since the low output probabilities due to the high-fan out in this model (up to 30 output nodes) are potentially at risk due to the relatively low 8-bit precision of Loihi's random number generator.

4.2. Supplementary Note 3: Additional Information on Main Text Examples

This note provides additional context and information for the example problems considered in the main text. The primary purpose of this section is to provide the equations and parameter values used along with any relevant discussion on building the particular discrete-time, discrete-space Markov chains associated with each example. Examples appear in the same order as the main text.

4.2.1. Particle Transport

In this section we will discuss further details of the particle transport examples in the main text. We first begin with some notation.

Assume that a particle occupies a single point with no mass and that intra-particle interactions can be ignored. The position and velocity of a particle is given by

$$r = (x_1,x_2,x_3) \quad and \quad v = v\Omega = v(sin\theta cos\phi, sin\theta sin\phi, cos\theta),$$

respectively. The quantity v is the particle speed. The angular density of particles at position r traveling in direction Ω with energy E at time t is denoted by $N(t,r,\Omega,E)$.

A quantity of interest when considering particle transport is the angular flux density, given by

$$\Phi(t,r,\Omega,E) = vN(t,r,\Omega,E).$$

We will assume that:

- energy E remains constant;
- we only care about either a single dimension or a projection into a single dimension, so that we may write $r = x$ and $\Omega = cos\theta$, or similar.

Under these assumptions, $\Phi := \Phi(t,x,\Omega)$. The angular flux density is then assumed to satisfy the Boltzmann particle transport equation given by

$$\frac{1}{v}\frac{\partial \Phi}{\partial t}(t,x,\Omega) + \Omega\frac{\partial \Phi}{\partial x}(t,x,\Omega) + (\sigma_s + \sigma_a)\Phi(t,x,\Omega) = \int \Phi(t,x,\Omega')\sigma_s(t,x,\Omega')p(\Omega' \to \Omega)d\Omega' + S(t,x,\Omega),$$

where σ_s and σ_a are functions representing the rates of particle scattering and absorption, and S is a function representing a particle source (Dupree and Fraley 2012). Additionally, the term $p(\Omega' \to \Omega)$ is the probability of transitioning from the direction Ω' to direction Ω after a scattering event; the arrow is not meant to indicate a limit.

To construct the equation used in the text, we further assume that the absorption and scattering rates do not depend on the direction of travel. Using the change of variables $\omega = \Omega' - \Omega$, the PIDE [eq:original_transport] can be written as

$$\frac{\partial \Phi}{\partial t}(t,x,\Omega) = -v\Omega\frac{\partial \Phi}{\partial x}(t,x,\Omega) - v\sigma_a(t,x)\Phi(t,x,\Omega) + vS(t,x,\Omega) + v\sigma_s(t,x)\int (\Phi(t,x,\Omega+\omega) - \Phi(t,x,\Omega))p(\omega \to 0 \mid \Omega)\, d\omega.$$

Note that the distribution on the change in direction ω depends on the direction before scattering Ω. This equation is the base equation used for the particle transport examples we consider.

4.2.2. Example 1: Simplified Transport

The first example we consider is an angular flux density problem that only depends on direction Ω and not space x. Essentially we consider a hypothetical particle that has two states: state 1 with $\Omega = 1$ and state 2 with $\Omega = -1$. The particle is subject to scattering and absorption events controlled by the constants σ_s and σ_a respectively. After scattering, the particle changes from state i to state j with probability p_{ij}. Note that p_{ii} or p_{jj} represents the probability that a particle does not change its state on a scattering event.

Since Φ does not depend on x in this example, the parameter v serves only to scale the absorption and scattering rates. Hence, we set $v = 1$ for clarity. Administering an initial condition, we seek to solve the PIDE

$$\frac{\partial}{\partial t}\Phi(t,\Omega) = -\sigma_a\Phi(t,\Omega) + \sigma_s\int (\Phi(t,\Omega+\omega) - \Phi(t,\Omega))p(\omega \to 0)\, d\omega$$
$$\Phi(0,\Omega) = g(\Omega) = \begin{cases} 5 & if\ \Omega = 1, \\ 3 & if\ \Omega = -1. \end{cases}$$

If we assume that $p_{ij} = 1/2$ for all i,j, then the analytic solution is given by

$$\Phi(t,\Omega) = \begin{cases} \frac{5}{2}\left(e^{-\sigma_a t} + e^{-(\sigma_a+\sigma_s)t}\right) + \frac{3}{2}\left(e^{-\sigma_a t} - e^{-(\sigma_a+\sigma_s)t}\right) & if\ \Omega = 1, \\ \frac{5}{2}\left(e^{-\sigma_a t} - e^{-(\sigma_a+\sigma_s)t}\right) + \frac{3}{2}\left(e^{-\sigma_a t} + e^{-(\sigma_a+\sigma_s)t}\right) & if\ \Omega = -1. \end{cases}$$

For the purposes of this example, we have assumed that $\sigma_a = 0.5$ and $\sigma_s = 5.0$.

The probabilistic representation gives

$$\begin{aligned} \Phi(t,\Omega) &= E\left[e^{-\sigma_a t} g(Y(t)) \mid Y(0) = \Omega\right], \\ dY(t) &= \omega_{Y(t)} dP(t). \end{aligned}$$

The stochastic process $Y(t)$ is effectively a proxy for our hypothetical particle. The particle retains its state (either 1 or -1) until the Poisson process $P(t)$ fires. Once the process fires, $Y(t)$ increases by the random change in direction $\omega_{Y(t)}$. We have included the subscript $Y(t)$ on ω to draw attention to the fact that the change in direction *depends* on direction before the scattering event. Notice, however, that our proxy process does not involve absorption. Rather, absorption is handled by the exponential term inside the expectation. This is an important observation since it shows that the underlying stochastic process may mimic physical processes, but it does not need to be identical to a physical process.

In order to develop the discrete-time Markov chain used to approximate the process $Y(t)$, we must first discuss the state space. For this problem, the state space is merely ± 1, so we do not need to discretize by choosing some increment $\Delta\Omega$. The next step in creation of the Markov chain is selecting an appropriate time discretization Δt.

Note that for any t and any Ω, the parameter of the Poisson process $P(t)$ is

$$\int_t^{t+\Delta t} \sigma_s du = \sigma_s \Delta t.$$

Ergo, our selection of Δt is independent of both t and Ω. In particular, we will want to choose Δt sufficiently small so that we can be reasonably sure that the Poisson process $P(t)$ will not fire more than one time in the time window Δt. Since the probability of no events occurring during the time window Δt is

$$q_0 = e^{-\sigma_s \Delta t},$$

and the probability of one event occurring is

$$q_1 = \sigma_s \Delta t e^{-\sigma_s \Delta t},$$

we want to choose Δt so that

$$q_{>1} = 1 - q_0 - q_1$$

is sufficiently small, or less than 0.05. Given the value of σ_s, a selection of $\Delta t = 0.01$ causes $q_{>1} \approx 0.001$.

For the construction of our Markov chain, we use q_1 for the probability that a scattering event occurs and $1 - q_1$ for the probability that no scattering event occurs. Technically, the stochastic differential equation representation for $Y(t)$ would have us calculate the probability for all increments in direction given the current direction and assign these probabilities to the states they would ultimately transition to. However, it is equivalent to instead change direction based on the given probabilities p_{ij}. Hence, we can define our Markov chain by the transition matrix

$$C = \begin{bmatrix} p_{11}q_1 + (1 - q_1) & p_{12}q_1 \\ p_{21}q_1 & p_{22}q_1 + (1 - q_1) \end{bmatrix}.$$

This transition matrix was used to inform a random walk process on TrueNorth. As shown in **Fig. 3c** and **Fig. 3d** in the main text, 1000 and 10000 random walks per starting location were sampled, respectively. These were averaged according to [eq:ivp_approx] to produce the curves shown. The analytic solution is also plotted for comparison.

The low bit resolution of the PRNG for the stochastic configuration used for our implementation forced the transition probabilities to be quantized to an 8-bit resolution. For this problem, with our defined parameters $C \approx \begin{bmatrix} 0.976219264387482 & 0.0237807356125179 \\ 0.0237807356125179 & 0.976219264387482 \end{bmatrix}$. However, the actual transition probabilities used by the TrueNorth implementation are

$$C \approx \begin{bmatrix} 0.9765625 & 0.0234375 \\ 0.0234375 & 0.9765625 \end{bmatrix} = \frac{1}{256}\begin{bmatrix} 250 & 6 \\ 6 & 250 \end{bmatrix}.$$

The simulation was run for $5.5\text{E} + 6$ hardware ticks. This was to ensure at least 500 simulation steps were produced, equating to a real simulation time of 5 seconds. The images shown in **Fig. 3c-d** in the main text are zoomed to show $t \in [0,2]$. Due to the random nature of the simulation, it is impossible to know, *a priori* exactly how many hardware ticks are needed in order to obtain exactly 500 simulation steps. Hence we must over estimate the run time and manually terminate.

4.2.3. Example 2: Particle Angular Fluence

The second particle transport example we consider pertains to angular fluence, or time-integrated flux. Up to changes in units, the angular fluence problem is given by [eq:transport_change_variables], but with the time derivative set to zero and no dependence on time. For our simplified example, we will assume that σ_s is a constant and that σ_a is zero. That is, there are no absorption events. We additionally assume that after scattering events, the new direction of particles is uniform on $[-1,1]$, and that we are concerned with $x \in [-1,1]$ only. Finally, we assume that the source term does not depend on direction of travel. Taken with an absorbing boundary condition, this gives the following problem:

$$\begin{aligned} 0 &= -v\Omega\frac{\partial}{\partial x}\Phi(x,\Omega) + vS(x) \\ &= +v\sigma_s\int(\Phi(x,\Omega+\omega) - \Phi(x,\Omega))p(\omega \to 0 \mid \Omega)d\omega, \quad x \in (-1,1), \Omega \in [-1,1], \\ \Phi(1,\Omega) &= 0, \quad if\ \Omega < 0, \\ \Phi(-1,\Omega) &= 0, \quad if\ \Omega > 0. \end{aligned}$$

For this example, we take $v = 200$, $\sigma_s = 0.15$, and

$$S(x) = \begin{cases} 3 & if\ |x| < 0.5 \\ 0 & otherwise. \end{cases}$$

There is no readily available analytic solution to this problem. The probabilistic solution is

$$\begin{aligned}\Phi(x,\Omega) &= E\left[\int_0^{T_{x,\Omega}} S(X(u))\,du \mid X(0)=x, Y(0)=\Omega\right],\\ dX(t) &= -vY(t)dt,\\ dY(t) &= \omega_{Y(t)}dP(t),\\ T_{x,\Omega} &= inf\{t>0 \mid, X(t)\notin[-1,1], X(0)=x, Y(0)=\Omega\}.\end{aligned}$$

Once again the stochastic process provides a proxy for our particle. The position $X(t)$ decreases according to its velocity (speed v times current travel direction $Y(t)$), and the current direction of travel $Y(t)$ is updated by the random increment $\omega_{Y(t)}$ every time the Poisson process fires. Again, this doesn't quite line up with the physics of the actual particle since the negative sign means our particle travels in the opposite direction.

To create the Markov chain approximation, we will need to discretize the state space. This involves choosing a Δx and $\Delta\Omega$ to create bins across the domain. For reasons that will become clear, we will wait to pick Δx until after we have chosen Δt. To begin, we select $\Delta\Omega = 1/15$. This yields 30 possible locations for $Y(t)$ in $[-1,1]$. From left to right, these values begin with $-1+1/30$ and end at $1-1/30$, increasing by $\Delta\Omega$. We will write $\Omega_j = -1+j\Delta\Omega$ for $j \in \{1,\ldots,30\}$.

Next we will choose a value for Δt. As in the previous example, for any time window $[t, t+\Delta t]$, the parameter for the Poisson process $P(t)$ is $\sigma_s\Delta t$. Using the same notation from the previous example for q_0, q_1, and $q_{>1}$, choosing $\Delta t = 0.01$ puts $q_{>1} \ll 0.05$.

Selecting both Δt and $\Delta\Omega$ will help us choose a Δx that will complement the problem. Note that in a single time window, a particle starting at some position (x,Ω) can only increment its position by $v\Omega\Delta t$. Once we discretized both direction and time, we have quantized the jumps the position can make based on the magnitude of the smallest nonzero direction. That is,

$$\Delta x = v\Delta\Omega\Delta t = \frac{1}{15}.$$

This yields 30 possible spatial locations. Again, we will denote these locations by x_j where $x_j = -1+j\Delta x$. These divisions yield a state space of size $30\times 30 = 900$.

Letting (i,j) represent the location (x_i,Ω_j), we now seek to calculate the transition matrix

$$C* = (c_{(i,j)\to(k,l)}),$$

where $c_{(i,j)\to(k,l)}$ represents the probability of transition from (i,j) to (k,l). We will assume some sort of ordering on the pairs (i,j) so that C^* is a 900×900 matrix. As implied in the previous discussion, the position x_i can only transition to $x_i - \Delta x\, sign(\Omega)$. As with the previous example, it is equivalent to choose our new directions based on the final specified distributions of directions rather than calculating the conditional density. In this example, we assumed directions after scattering are uniform. Therefore new directions in our discretized space are selected with probability $1/30$. Hence

$$c_{(i,j)\to(k,l)} = \begin{cases} (1-q_1) + \dfrac{q_1}{30} & if\ x_k = x_i + \Delta x\, sign(\Omega_j)\ and\ j = l, \\ \dfrac{q_1}{30} & if\ x_k = x_i + \Delta x\, sign(\Omega_j)\ and\ j \neq l, \\ 0 & otherwise. \end{cases}$$

This, however, does not define the entire transition matrix. We have not accounted for random walks that would travel out of the domain. This occurs when the transition from the state (i,j) would create an x value that no longer falls in $[-1,1]$. To this end, we create an additional state, say a that corresponds to absorption. We must now define the column vector of probabilities

$c_a = (c_{(i,j)\to a})$.

The probability of transition to this state is simply 1 whenever the increment would force the walk to exit the domain. There is no need to keep track of direction at this point. Hence,

$$c_{(i,j)\to a} = \begin{cases} 1 & if\ |x_i + \Delta x\, sign(\Omega_j)| > 1 \\ 0 & otherwise. \end{cases}$$

Finally, allowing $\bar{0}$ to represent a row vector of 900 zeros, the full 901×901 transition matrix for the Markov chain can be written as

$$C = \begin{pmatrix} C^* & c_a \\ \bar{0} & 1 \end{pmatrix}.$$

This transition matrix was used to implement 6250 random walks per possible starting location on Loihi. Random walks are allowed to run until they arrive at the absorption state. The data from the collected random walks is averaged in the Monte Carlo sense via equation [eq:fluence_sol]. We note that the averaging requires the use of a Riemann sum approximation to an integral. However, rather than keep information on every individual path, the Riemann sum term can be collapsed using cumulative densities. For more information on this technique, see [34]. The result is plotted as Figure 11**g** in the main text. For comparison, a simulation performed on MATLAB for 1 million walkers per location is shown in Figure 11**h**.

4.3. Neuromorphic approach to simulating on non-Euclidean geometries

The particle examples above are straightforward demonstrations of RWs with non-local jumps on a simple domain. We next demonstrated neuromorphic RWs over non-Euclidean domains, solving two heat equations. By carefully defining a mesh and calculating transition probabilities, PIDEs over large complex geometries are no problem for the neuromorphic RW method. To demonstrate the ability of this method to solve problems on non-Euclidean domains, we present two examples involving spheres. While the non-Euclidean shapes we consider are by no means 'complex,' we merely showcase that the method is mostly agnostic to the domain.

Consider a basic heat equation on the unit sphere. We let S^2 represent the unit sphere and take $a(t,x)$ to be the positive scalar α for all $t \in [0,\infty)$ and $x \in S^2$. Set the remaining coefficients in Equation 4 to zero. Paired with an initial condition $g(x)$, the probabilistic solution to the heat equation on the sphere is given by

$$u(t,x) = E[g(X(t))|X(0) = x],$$
$$dX(t) = \sqrt{2\alpha}dW(t),$$

Equation 10

where $W(t)$ represents Brownian motion on the surface of the sphere. By choosing a particular initial condition, this problem has a tractable analytic solution (**[SN3]**). The initial condition selected resembles a soccer ball pattern (Figure 12**a**).

To employ our neuromorphic approach, we must be able to approximate Brownian motion on the surface of the sphere with a DTMC. There are several ways to describe Brownian motion on the sphere, including using the von-Mises Fisher distribution [37], employing representations with spherical coordinates [7], or limiting from higher dimensions [8]. Since it is applicable to other curved shapes, we elect to use a tangent plane approximation. Setting $\alpha = 42$, we deploy a RW approximating the process $X(t)$ on Intel's Loihi platform. Starting 4000 RWs on each position yields the approximate solution found in Figure 12**a** for a collection of time points. We would expect better agreement with a greater number of nodes and walkers per starting location. Further, as shown in Figure 12**b**, we see that the low precision of probability transition has acutely increased the amount of error accrued for this example when compared to a MATLAB simulation.

This example provides compelling evidence that complex geometries where analytic methods are less tractable represent an opportunity for NMC impact. These could arise where domains are more complicated than just a single sphere. One could imagine an object with many spines or with several crevices. To start down this path, we present an initial-value problem on the surface of a barbell shaped object.

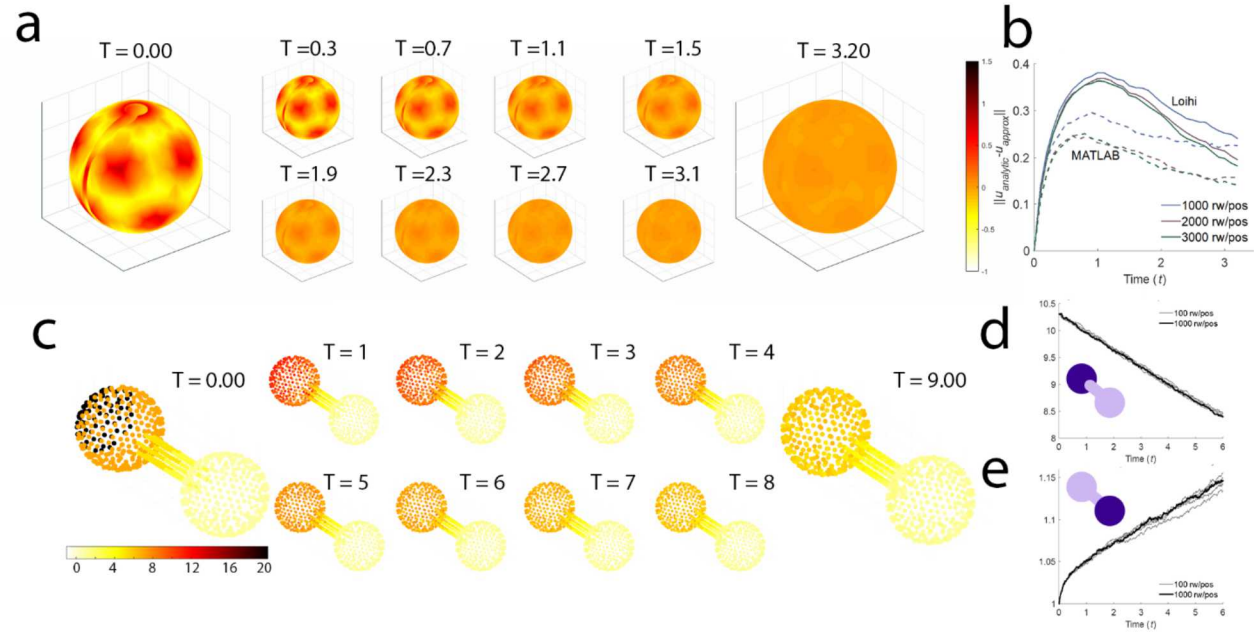


Figure 12: NMC random walk algorithm can implement random walks over non-Euclidean geometries. (a) Time-course of random walks simulated on Loihi to model heat diffusion on the surface of a sphere. Red locations represent higher initial temperature relative to yellow locations.

Heat is conserved on this simulation. (b) Absolute norm of error is higher on NMC relative to MATLAB simulation at initial timepoints, but approaches conventional error levels as simulation progresses. (c) Time-course of random walks run on neural simulator for heat diffusion on two spheres connected by a tube ("barbell"). Heat was allowed to dissipate from the surface. (d) Average temperature of the left sphere decreases rapidly during the simulation. (e) Temperature gradually increases sharply for small time on the right sphere. As time increases, this rate of increase slows as cooling begins to take effect. For large time, the temperature on the right sphere will decrease to zero.

Consider the heat flow on a barbell with cooling and an initial condition. Let B represent the surface of the barbell shape. Again, we set $a(t,x) = \alpha$, some positive scalar. Then, to account for cooling, we take $c(t,x) = \kappa$, another positive scalar. All other coefficients in Equation 4 are assumed to be zero. Again letting $g(x)$ be an initial condition, the probabilistic solution is

$$u(t,x) = E\left[e^{-\kappa t} g(X(t)) \middle| X(0) = x\right],$$

$$dX(t) = \sqrt{2\alpha}\, dW(t),$$

Equation 11

where $W(t)$ now represents Brownian motion on B. Our discretization of the shape required 748 mesh points (more details on the mesh construction and DTMC are in **[SN3]**). Due to the mesh-size relative to the currently limited neuromorphic chip sizes available to us, we deployed this example on a spiking net simulator. We implemented a random walk approximating the stochastic process. The results of simulation for various time points can be found in Figure 12**c**.

4.4. Non-Euclidean Geometries

The two previous examples utilize random walks on a domain that is not very complicated. The method does extend to more complicated domains, although there is some nuance in execution. Any time a diffusion coefficient exists ($a \neq 0$), the underlying SDE contains a Brownian motion term ($W(t)$). As detailed in **Supplementary Note 2**, this is a Brownian motion with respect to the appropriate probability space. If the problem were in R^d, then $W(t)$ represents a standard d-dimensional Brownian motion.

On the other hand, if the problem were on the surface of some 3-dimensional shape, $W(t)$ is not a 3-dimensional Brownian motion, but rather a Brownian motion on the surface of the shape. For smooth shapes, this means that locally $W(t)$ is a 2-dimensional Brownian motion, however more complicated distributions can be defined (see (Watson 1982) for a discussion on the von Mises-Fisher distribution, a distribution describing Brownian motion on the surface of the sphere).

In this section we consider two examples monitoring heat transport on the surface of non-Euclidean objects. The first is a sphere; a smooth shape where locally the diffusion is a 2-dimensional Brownian motion. Here, transition probabilities are calculated by projecting to a tangent plane. The second is two spheres joined by a hexagonal prism. The shape is not smooth where the prism joins the spheres and along the spines of the prism. On the sphere, the same tangent plane approximation is used. On and near the prism, an unfolding argument is applied. See Figure 13 for a visualization of the sphere and barbell.

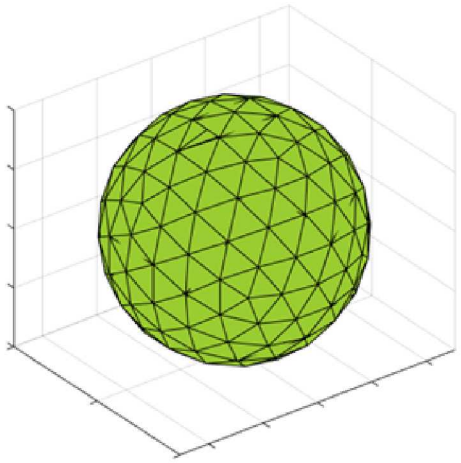


Figure 13: Sphere Mesh Structure

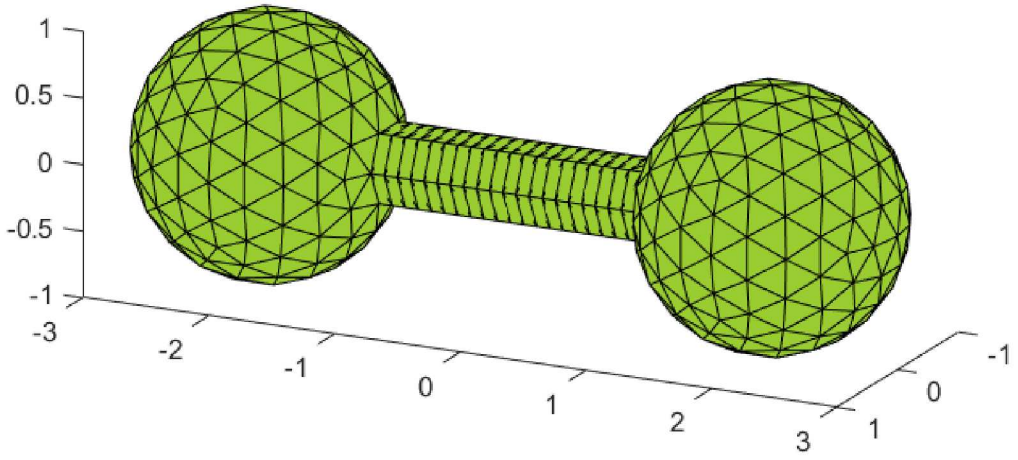


Figure 14: Barbell Mesh Structure

4.4.1. Heat Equation on the Surface of the Unit Sphere

Let S^2 represent the unit sphere and consider the initial value heat equation:

$$\frac{\partial}{\partial t}u(t,x) = \alpha\nabla_x u(t,x), \quad x \in S^2,\ t \in (0,\infty),$$
$$u(0,x) = g(x).$$

With an appropriate g, the probabilistic solution is

$$u(t,x) = E[g(X(t)) \mid X(0) = x]$$
$$dX(t) = \sqrt{2\alpha}dW(t),$$

where $W(t)$ is a Brownian motion on the surface of the sphere. Let Y_m^n represent the real part of the spherical harmonic of degree m and order n. Suppose

$$g(x) := g(\theta,\phi) = Y_6^0(\theta,\phi) + \sqrt{\frac{14}{11}}Y_6^5(\theta,\phi),$$

where (θ,ϕ) is the spherical coordinate for the vector x on the unit sphere with $\theta \in [0,\phi]$ and $\phi \in (-\pi,\pi]$. That is, θ measures the angle from the north pole and ϕ measures the angle from the prime meridian. Since the spherical harmonics are the eigenfunctions of the Laplacian in three dimensions, this initial condition admits a tidy analytic solution to [eq:sphere_heat_1]:

$$u(x) := u(\theta,\phi) = e^{-42\alpha t}g(\theta,\phi).$$

Let $\alpha = 1/42$ be given.

In order to perform our random walk approximation, we will first need to discretize the sphere. In the previous examples we took a uniform approach to discretization. In this example we will create roughly congruent triangles on the surface of the sphere. This is accomplished by calculating a geodesic dome structure.

Begin with an icosahdron with vertices on the surface of the unit sphere. An icosahedron is a polyhedron constructed with 20 equilateral triangles and 12 vertices. Our construction begins with two of these vertices equidistant from the north pole (and consequently two others equidistant from the south pole). At the midpoint of each triangle, a new line is drawn, creating four new equilateral triangles on each face of the icosahedron. Once more, on the midpoint of each triangle a new line is drawn creating four new smaller triangles. This yields 16 small equilateral triangles on each face of the original icosahedron for a total of 320 triangles. The vertices of these triangles are projected to the surface of the sphere (see Figure 13).

Our random walk on the sphere will traverse over the centroids of these triangles, projected to the surface of the sphere. We will allow the random walk to transition to any triangle neighbor that shares a vertex and back to itself. Since the original icosahedron had 12 vertices, this means that 60 triangles with have 12 possible transitions and the remaining 260 will have 13 possible transitions.

We must now select a time step size Δt and compute transition probabilities for the triangles. We will ultimately use the tangent plane approximation method for this calculation. However, some may wonder why not use spherical coordinates since our initial condition and analytic solution utilize these coordinates. Translating [eq:sphere_heat_1] into spherical coordinates yields

$$\begin{aligned} \frac{\partial}{\partial t}u(t,\theta,\phi) &= \alpha\left(\frac{\partial^2}{\partial\theta^2}u(t,\theta,\phi) + cot\theta\frac{\partial}{\partial\theta}u(t,\theta,\phi) + csc^2\theta\frac{\partial^2}{\partial\phi^2}u(t,\theta,\phi)\right), \\ u(0,\theta,\phi) &= g(\theta,\phi). \end{aligned}$$

This gives the probabilistic solution

$$\begin{aligned} u(t,\theta,\phi) &= E[g(X(t),Y(t)) \mid X(0) = \theta, Y(0) = \phi] \\ dX(t) &= \alpha cot X(t) + \sqrt{2\alpha}dW_1(t) \\ dY(t) &= \sqrt{2\alpha csc^2 X(t)}dW_2(t). \end{aligned}$$

We can already see an issue with taking this approach – when $X(t)$ nears the north or south pole, the drift in latitude approaches infinity and the diffusion in longitude approaches infinity. We do not take this approach to avoid dealing with this singularity.

Order the 320 triangles in some fashion. Given a current triangle center $r_i = (x_i, y_i, z_i)$, we project to a tangent plane using the gnomonic projection. This projects points to a tangent plane so that arcs along a great circle are projected onto straight line segments. We accomplish this by rotating the sphere so that r_i is at the north pole $(0,0,1)$ and then projecting all neighboring triangles to the xy tangent plane centered at $(0,0)$. A standard two dimensional Brownian motion is used to approximate the probability of transition into each of these projected triangles. Under such a rotation and projection, the new coordinates (x',y') of any point (x,y,z) in terms of (x_i, y_i, z_i) are given by

$$(x',y') = \begin{cases} \left(\frac{x}{z}, \frac{y}{z}\right) & \text{if } x_i^2 + y_i^2 = 0 \\ \left(\frac{z_i(x_i x + y_i y) - z\left(x_i^2 + y_i^2\right)}{\sqrt{x_i^2 + y_i^2}(x_i x + y_i y + z_i z)}, \frac{x_i y - y_i x}{\sqrt{x_i^2 + y_i^2}(x_i x + y_i y + z_i z)}\right) & \text{otherwise.} \end{cases}$$

We note that in the construction we have detailed here, the north and south poles are always a vertex of a triangle and are never a center point. Therefore, the first assignment in [eq:gnomonic_projection] is never used.

Suppose that r_i and r_j are centers of triangles that share a vertex. We use the SDE for $X(t)$ (see [eq:prob_sphere]) to inform our calculation of transition probability from r_i to r_j. There is no drift term. Locally, $X(t)$ is a two-dimensional Brownian motion. Letting X' represent the projection to the tangent plane, then $X'(t)$ is Gaussian with mean $X'(0)$ and covariance matrix given by

$$\Sigma_t = \begin{bmatrix} 2\alpha t & 0 \\ 0 & 2\alpha t \end{bmatrix}.$$

Applying Euler-Maruyama, $X'(t + \Delta t)$ is Gaussian with mean $X'(t)$ and covariance matrix given by

$$\Sigma_{\Delta t} = \begin{bmatrix} 2\alpha \Delta t & 0 \\ 0 & 2\alpha \Delta t \end{bmatrix}.$$

Let τ_{ji} represent the triangle with center r_j in the tangent plane created by the gnomonic projection about the point r_i. Let $f_G(\rho,\mu,\Sigma)$ be the probability density function for the two dimensional Gaussian with mean μ and covariance matrix Σ at the point ρ. Recalling that the projection of r_i is $(0,0)$, then we approximate the probability of transition from r_i to r_j by

$$p_{ij} \approx \int_{\tau_{ji}} f_G(\rho,(0,0),\Sigma_{\Delta t})\, d\rho,$$

where $\Sigma_{\Delta t}$ is given by [eq:cov_matrix]. After selecting Δt, this integral can be evaluated numerically. We elected to use a Gaussian quadrature method. A selection of $\Delta t = 0.1$ ensures that the probability of transition to any triangle outside of those triangles that share a vertex with the starting location is less than 0.05. Departing from the previous examples, we added the missing probability for transition to the probability of not changing locations. That is, we added $1 - \sum_j p_{ij}$ to the probability p_{ii}. This is a simplifying choice we have made. Integration of transition to any other possible triangle

via [eq:sphere_transition] is not possible as the approximation is only valid locally and also because the gnomonic projection does not work for triangles in the opposite hemisphere.

Setting our transition matrix to

$C = (p_{ij})$,

we implemented a graph for random walks on Loihi over the 320 possible locations. Using 3000 walkers total, the solution was calculated by changing the center points to spherical coordinates and averaging in the Monte Carlo sense via equation [eq:prob_sphere]. The simulation result displayed for various frames in time is given in **Fig. 4a** in the main text. Additionally, the norm of the difference in the Loihi calculated solution and the analytic solution over time is plotted in **Fig. 4b**.

4.4.2. Heat Transport on the Surface of a Barbell

Our final example is heat transport on a barbell shape. The barbell in consideration is two unit spheres joined by a hexagonal prism (see Figure 14). We created our barbell shape by starting with two unit spheres, centered on ± 2. When performing the triangular mesh construction on the sphere as in the previous example, there will exist a left- or right-most vertex, closest to zero for each sphere. This vertex will belong to six triangles. We replace these triangles on either side with a hexagonal prism with side lengths equal to the length of the hexagon sides formed by the six triangles. Since the vertex we use to make this replacement comes from an initial division in the construction, this is a regular hexagon.

Let B be the surface of the two unit spheres joined by the described hexagonal prism. We are interested in solving

$$\begin{aligned} \frac{\partial}{\partial t}u(t,x) &= \alpha\nabla_x u(t,x) - \kappa u(t,x), \qquad x \in B, t \in (0,\infty) \\ u(0,x) &= g(x). \end{aligned}$$

The parameter κ can be thought of as a rate of cooling. For this problem, we assume that $\alpha = 1/2$, $\kappa = 0.05$. Letting $x = (x,y,z)$, we take

$$g(x) = \begin{cases} 20 & y \geq 2.5, \\ 7 & 2.5 > y \geq 1, \\ 5 & 1 > y \geq 0, \\ 3 & 0 > y \geq -1, \\ 1 & -1 > y. \end{cases}$$

The probabilistic solution is given by

$$\begin{aligned} u(t,x) &= E\left[e^{-\kappa t} g(X(t)) \mid X(0) = x\right] \\ dX(t) &= \sqrt{2\alpha}\, dW(t), \end{aligned}$$

where $W(t)$ represents a Brownian motion on the surface of the barbell.

We discretize the spheres in the same manner from the previous example. We divide the hexagon into rectangles as follows. All the triangles on the sphere have roughly equal area by construction. We select one of the triangles with the smallest area, one of the triangles that shares a vertex with the original vertices of the icosahedron. Using this area and the length of the edge of the hexagonal prism, we determine the ideal width of a rectangle to equal the area of this triangle. We then round this width based on the closest number of rectangles we can place along the prism.

From this construction, we get 314 triangles for each sphere and 120 rectangles in the prism, for a total of 748 locations in the state space. The states are taken to be the centroids of the triangles, projected to the surface of the sphere, and the centers of the 120 rectangles.

Probabilities of transition among triangles on the left and right spheres are handled like in the previous example. Triangles are again considered adjacent if they share a vertex. Transitions from a triangle on the sphere to a rectangle on the prism are only allowed if the rectangle shares a vertex with the triangle. Since there are six rectangles replacing six triangles on each sphere, and these rectangles are roughly equal in area to the triangles, we assign the probability of transition from a triangle to an adjacent rectangle to be the probability of transition from the triangle to the triangle that the rectangle is replacing. Again, this is not perfect. This is a choice we have made. The tangent plane projection does not work as well in these locations because the rectangles on the prism are almost perpendicular to the tangent plane.

Transitions among rectangles on the prism are allowed to other rectangles that share at least one vertex and back to the original rectangle. Transitions are calculated by unfolding the prism, setting the center of the rectangle equal to $(0,0)$, and calculating the probability of transitioning into the rectangles surrounding the current location. This probability is calculated via [eq:sphere_transition], where the integration is performed over the appropriate rectangle rather than triangle.

Transitions from a rectangle on the prism to a triangle on the sphere are handled similarly. First, the triangle sharing a side with the rectangle is unfolded into the plane with the center of the rectangle occupying $(0,0)$. The probability of transition into this triangle is calculated via [eq:sphere_transition]. Now, depending on the rectangle (transitioning to left or right sphere), there are six additional triangles that share a vertex. These share the upper and lower vertices on one side of the rectangle, dependent on whether the rectangle is transitioning to the right or left sphere. There are three for the upper vertex and three for the lower vertex. We approximate the transition to one of these three upper triangles by calculating the probability of moving into the entire quarter plane diagonal from the upper vertex. This is accomplished by replacing the integration bounds in [eq:sphere_transition] with the appropriate bounds for the quarter plane. This probability is divided by 3 and assigned to each of the three triangles sharing a vertex. Again, this is a simplifying choice as unfolding the three triangles is difficult in this scenario. This is repeated for the other three triangles touching the lower vertex.

Through this calculation, all possible transitions are calculated for locations that share a vertex. By selecting $\Delta t = 0.005$, we ensured that the probability of transition outside the allowable locations for each starting location in the state space was less than 0.05. As in the previous example, we add any missing probability to the probability of transitioning to the same location to ensure a total probability.

We use this to define a transition matrix. The graph for this random walk was implemented on a spiking net simulator as in (Smith et al. 2020). Starting 1000 walkers on each location and averaging in the Mote Carlo sense via [eq:barbell_prob_sol], we calculated an approximate solution, plotted in Figure 12 above.

5. DISCUSSION

The results here demonstrate that spiking neuromorphic hardware technology is suitable for scaling to an energy-efficient approach to solving an important set of numerical computing problems. Neuromorphic hardware is still immature relative to conventional hardware in terms of both physical scale and clock speed, although it already demonstrates considerable power advantages. Here, we show that our neural RW algorithm scales comparably to a parallel CPU approach, allowing us to observe a significant energy advantage in current neuromorphic platforms today while being positioned to take full advantage of large-scale neuromorphic hardware once practical. We further focus our exploration on demonstrating the broad application impact of our algorithm approach, showing that with simple extensions this approach can apply to a wide range of complex application domains.

Notably, the approach taken here does not leverage all the brain-inspired features present in many emerging neuromorphic hardware technologies. For instance, our approach does not leverage learning; however, we expect that the neural formulation of stochastic processes may make them more amenable to model calibration against experimental observations, and in situ neuromorphic learning may make this process more efficient. Likewise, we focused our demonstrations on large-scale digital spiking platforms, such as Loihi and TrueNorth, because they exist at the requisite neural scales for our algorithms. There is considerable interest in analog neuromorphic approaches that should similarly be compatible with this approach, although we would have to consider the precision implications of analog devices alongside the other approximation considerations (Figure 9).

One important consideration of this work is that the numerical accuracy of our neural approach is relatable to typical numerical precision considerations in conventional computing. Stated differently, this approach avoids the approximation pitfalls associated with many AI algorithms, wherein the implications of numerical precision and interpretability is still an open question. While understanding and accounting for these approximation errors will be critical for any application, the graph-based approach taken here provides a number of well-understood design choices to tailor the algorithm and hardware solution appropriately given precision concerns. For instance, in applications with clearly defined state spaces, such as diffusion over a social network, the mesh can directly map to the system, and resources can be dedicated to adding more walkers. Alternatively, in complex geometries or unbounded systems, it may be necessary to commit considerable neuromorphic hardware to a larger mesh.

Whatever the eventual set of capabilities that future neuromorphic platforms have; we expect that neuromorphic hardware will eventually exist primarily in heterogeneous system architectures alongside CPUs, GPUs, and other accelerators [19]. The neuromorphic algorithms for solving PDEs described here complement AI as an application for brain-inspired hardware, and they strengthen the long-term value proposition for neuromorphic hardware in future computing systems. Further, in contrast to neural network applications, where neuromorphic hardware has struggled to match the speed of GPUs and linear algebra accelerators, our work shows that in the realm of numerical computing, neuromorphic hardware not only can deliver concrete energy advantages today, but is capable of scaling effectively in terms of processing time and overall efficiency.

REFERENCES

1. Agarwal, S., Quach, T.-T., Parekh, O., Hsia, A.H., DeBenedictis, E.P., James, C.D., Marinella, M.J. and Aimone, J.B. Energy scaling advantages of resistive memory crossbar based computation and its application to sparse coding. *Frontiers in neuroscience*, *9*. 484.
2. Aimone, J.B. Neural algorithms and computing beyond Moore's law. *Communications of the ACM*, *62* (4). 110-110.
3. Aimone, J.B., Hamilton, K.E., Mniszewski, S., Reeder, L., Schuman, C.D. and Severa, W.M., Non-neural network applications for spiking neuromorphic hardware. in *3rd International Workshop on Post-Moore's Era Supercomputing (PMES 2018), Dallas, TX*, (2018).
4. Aimone, J.B., Parekh, O., Phillips, C.A., Pinar, A., Severa, W. and Xu, H., Dynamic Programming with Spiking Neural Computing. in *Proceedings of the International Conference on Neuromorphic Systems*, (2019), ACM, 20.
5. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G.S.L., Buell, D.A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M.P., Hartmann, M.J., Ho, A., Hoffmann, M., Huang, T., Humble, T.S., Isakov, S.V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P.V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark, M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J.R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mohseni, M., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M.Y., Ostby, E., Petukhov, A., Platt, J.C., Quintana, C., Rieffel, E.G., Roushan, P., Rubin, N.C., Sank, D., Satzinger, K.J., Smelyanskiy, V., Sung, K.J., Trevithick, M.D., Vainsencher, A., Villalonga, B., White, T., Yao, Z.J., Yeh, P., Zalcman, A., Neven, H. and Martinis, J.M. Quantum supremacy using a programmable superconducting processor. *Nature*, *574* (7779). 505-510.
6. Bossy, M. and Champagnat, N. Markov processes and parabolic partial differential equations, 2010.
7. Brillinger, D.R. A Particle Migrating Randomly on a Sphere David R. Brillinger.
8. Carlsson, T., Ekholm, T. and Elvingson, C. Algorithm for generating a Brownian motion on a sphere. *Journal of physics A: Mathematical and theoretical*, *43* (50). 505001.
9. Chung, K.L., Williams, R.J. and Williams, R. *Introduction to stochastic integration*. Springer, 1990.
10. Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N. and Jain, S. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38* (1). 82-99.
11. Feynman, R.P. Quantum mechanical computers. *Foundations of physics*, *16* (6). 507-531.
12. Furber, S.B., Galluppi, F., Temple, S. and Plana, L.A. The spinnaker project. *Proceedings of the IEEE*, *102* (5). 652-665.
13. Garroni, M.G. and Menaldi, J.L. *Second order elliptic integro-differential problems*. CRC Press, 2002.
14. Grigoriu, M. *Stochastic calculus: applications in science and engineering*. Springer Science & Business Media, 2013.
15. Hamilton, S.P., Slattery, S.R. and Evans, T.M. Multigroup Monte Carlo on GPUs: Comparison of history-and event-based algorithms. *Annals of Nuclear Energy*, *113*. 506-518.
16. Hanson, F.B. *Applied stochastic processes and control for jump-diffusions: modeling, analysis and computation*. SIAM, 2007.

17. Harrow, A.W. and Montanaro, A. Quantum computational supremacy. *Nature*, *549* (7671). 203.
18. Konakov, V. and Mammen, E. Local limit theorems for transition densities of Markov chains converging to diffusions. *Probability theory and related fields*, *117* (4). 551-587.
19. Krichmar, J.L., Severa, W., Khan, S.M. and Olds, J.L. Making BREAD: Biomimetic strategies for artificial intelligence now and in the future. *Frontiers in neuroscience*, *13*. 666.
20. Lanyon, B.P., Whitfield, J.D., Gillett, G.G., Goggin, M.E., Almeida, M.P., Kassal, I., Biamonte, J.D., Mohseni, M., Powell, B.J. and Barbieri, M. Towards quantum chemistry on a quantum computer. *Nature chemistry*, *2* (2). 106.
21. Masuda, N., Porter, M.A. and Lambiotte, R. Random walks and diffusion on networks. *Physics reports*, *716*. 1-58.
22. McCulloch, W.S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5* (4). 115-133.
23. Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C. and Nakamura, Y. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345* (6197). 668-673.
24. Mniszewski, S.M., Graph Partitioning as Quadratic Unconstrained Binary Optimization (QUBO) on Spiking Neuromorphic Hardware. in *Proceedings of the International Conference on Neuromorphic Systems*, (2019), ACM, 4.
25. Parekh, O., Phillips, C.A., James, C.D. and Aimone, J.B., Constant-Depth and Subcubic-Size Threshold Circuits for Matrix Multiplication. in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, (2018), ACM, 67-76.
26. Protter, P. *Stochastic Integration and Differential Equations: A New Approach*. Springer, 1990.
27. Roy, K., Jaiswal, A. and Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, *575* (7784). 607-617.
28. Schuman, C.D., Hamilton, K., Mintz, T., Adnan, M.M., Ku, B.W., Lim, S.-K. and Rose, G.S., Shortest path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation. in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, (2019), ACM, 3.
29. Severa, W., Lehoucq, R., Parekh, O. and Aimone, J.B., Spiking neural algorithms for markov process random walk. in *2018 International Joint Conference on Neural Networks (IJCNN)*, (2018), IEEE, 1-8.
30. Severa, W., Vineyard, C.M., Dellana, R., Verzi, S.J. and Aimone, J.B. Training deep neural networks for binary communication with the Whetstone method. *Nature Machine Intelligence*, *1* (2). 86.
31. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, *41* (2). 303-332.
32. Siu, K.-Y., Roychowdhury, V. and Kailath, T. *Discrete neural computation: a theoretical foundation*. Prentice-Hall, Inc., 1995.
33. Skorokhod, A.V. *Studies in the theory of random processes*. Courier Dover Publications, 1982.
34. Smith, J.D., Severa, W., Hill, A.J., Reeder, L., Franke, B., Lehoucq, R.B., Parekh, O.D. and Aimone, J.B. Solving a steady-state PDE using spiking networks and neuromorphic hardware. *arXiv preprint arXiv:2005.10904*.
35. Tankov, P. *Financial modelling with jump processes*. CRC press, 2003.

36. Von Neumann, J. *The Computer and the Brain*. Yale University Press, 2000.
37. Watson, G.S. Distributions on the circle and sphere. *Journal of Applied Probability*. 265-280.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	01977	sanddocs@sandia.gov

This page left blank

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.