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

Neuromorphic computing, which aims to replicate the computational structure and architecture of the brain in synthetic hardware, typically focuses on artificial intelligence applications. What is less explored is whether such brain-inspired hardware can provide value beyond cognitive tasks. Here we show that the high degree of parallelism and configurability of spiking neuromorphic architectures makes them well suited to implement random walks via discrete-time Markov chains. These random walks are useful in Monte Carlo methods, which represent a fundamental computational tool for solving a wide range of numerical computing tasks. Using IBM’s TrueNorth and Intel’s Loihi neuromorphic computing platforms, we show that our neuromorphic computing algorithm for generating random walk approximations of diffusion offers advantages in energy-efficient computation compared with conventional approaches. We also show that our neuromorphic computing algorithm can be extended to more sophisticated jump-diffusion processes that are useful in a range of applications, including financial economics, particle physics and machine learning.

Despite the increasing ability to develop large-scale neural processors today, the theoretical value of neuromorphic hardware remains unclear—unlike quantum computing that offers clear fundamental advantages at scale. Nevertheless, there are several architectural features of most nervous systems that could yield advantages including the high degree of connectivity between neurons, the colocation of processing and memory, and the use of action potentials (referred to as spikes) to communicate. Algorithm research for spiking neuromorphic hardware has primarily focused on its suitability for deep learning and other emerging artificial intelligence (AI) algorithms. Such applications are straightforward, given the alignment of neural architectures with neural networks, and it can be expected that the value of neuromorphic computing will grow as AI algorithms derive further inspiration from the brain. However, the impact of neuromorphic computing beyond cognitive applications is less certain.

Quantum computing has shown how emerging hardware can have an impact beyond its original inspiration: it was conceived as a means for efficient chemistry simulations, but is now recognized as useful in a much broader range of applications. Unlike quantum computing, which faces technical challenges in scaling up, neuromorphic platforms can already be scaled to non-trivial sizes, with several multi-chip spiking neuromorphic systems achieving scales of over a hundred million neurons.

However, identifying neuromorphic computing value for any specific application is complicated because its main advantage is typically energy efficiency as opposed to faster computation (although speed benefits remain a possibility), and its technologies are immature compared with conventional von Neumann systems, which have been optimized over decades. We define an algorithm as having a neuromorphic advantage if that algorithm shows a demonstrable advantage compared with the von Neumann architecture in one resource (for example, energy) and exhibiting comparable or better scaling in other resources (for example, time). Because neuromorphic hardware currently offers advantages in power consumption, we focus on algorithms that show comparable or better time scaling compared with the von Neumann architecture and still requiring less total energy to perform the same computation.

Observing a neuromorphic advantage for non-cognitive applications should not be taken as a given since the specialization of computer architectures to improve performance on a subset of tasks will likely result in degraded performance in other tasks. Therefore, observing a neuromorphic advantage on non-cognitive applications would demonstrate that neuromorphic computing can have a broader impact than previously assumed and provide a concrete framework by which to develop the technology. Although a definitive neuromorphic advantage (as defined here) has not yet been demonstrated for non-cognitive applications, there are three categories of such computing tasks that appear well suited for neuromorphic computing: linear algebra, in which the high fan-in of neurons can be used to realize known theoretical advantages of threshold gate (TG) logic graph analytical tasks that can leverage the configurability and parallelization of neural circuits and sampling steady-state distributions for a wide range of potential applications using stochastic neural circuits.

In this Article, we show that large-scale neuromorphic hardware can offer a neuromorphic advantage 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, a process often referred to as Monte Carlo. Diffusion is a typical component of the underlying SDEs used in the probabilistic solution of the PIDEs. We can show our neuromorphic computing algorithm for generating random walk approximations to diffusion satisfies our neuromorphic advantage criteria on two current large-scale neuromorphic platforms: the IBM neurosynaptic system (known as TrueNorth) and the Intel Loihi system. Although these are distinct neural architectures, they both directly implement a large number of neurons in silicon and are readily scalable to multi-chip platforms. We also show that our neuromorphic random walk algorithm can be extended to account...
for more sophisticated jump-diffusion processes that are useful for addressing a wide range of applications, including financial economics (such as option pricing models), particle physics (such as radiation transport) and machine learning (such as diffusion maps).

**Neuromorphic advantage on simulating random walks**

Random walk solutions are often an attractive option for large-scale simulation of stochastic processes. Although deterministic numerical solutions of PIDEs often rely on relatively few large complex calculations, random walks typically rely on many simple computations. These computations can be efficiently implemented within circuits of spiking neurons.

We extend a neuromorphic circuit to model random walks in diffusion to efficiently solve PIDEs. This algorithm represents a stochastic process as a mesh, with each mesh point consisting of two

---

**Fig. 1** Neuromorphic hardware can efficiently implement random walks. 

- **a.** Leaky IF 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 between 10 and 20 leaky IF 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 neuromorphic computing algorithm implements both stochastic and deterministic state transitions of all the random walkers at all the mesh points in parallel.  
- **d.** Demonstration of simple diffusion on a 30 × 30 torus on the Intel Loihi platform. Aside from reading out intermediate states for visualization, the entire random walk process was performed within the neuromorphic computing system.  
- **e.** Simulating additional walkers on Intel Loihi and IBM TrueNorth increases time efficiency, whereas additional walkers have the same cost on a single CPU core.  
- **f.** Random walk processes can be distributed over multiple meshes on IBM TrueNorth or multiple cores on CPUs and GPUs. GPUs are considerably faster due to their large number of parallel processes.  
- **g.** Compared with the relative cost of simulating 1,000 walkers, time-scaling on the parallel neuromorphic computing implementation on TrueNorth matches multi-threaded CPUs and GPUs, and single-chip scaling is considerably better on neuromorphic computing platforms.  
- **h.** Neuromorphic computing platforms, TrueNorth and Loihi, have a considerably higher energy efficiency (walker updates per joule) than CPUs and GPUs. The legend is the same as **g.**  
- **i.** Trade space of energy and time costs for walker updates in neuromorphic computing and conventional platforms. The solid lines represent single-thread implementations, and the dashed lines represent multi-threaded implementations. All the scaling experiments have ten replicates with standard errors below 0.5%; therefore, the error bars are not shown.

---

**References**

26. Nature Electronics [URL](https://www.nature.com/natureelectronics)
**Fig. 2** | Random walk processes are well suited for neuromorphic computing, and the inclusion of different terms in the stochastic process yields random walks with differing behaviours. **a–e**, Left: three illustrative random walks for 100 time steps. Right: density of 1,000 random walkers run on Loihi. The range shown in the density plots is highlighted and overlaid onto the process examples. The changing terms \( a, b, c, h \) and \( \lambda \) are found in equation (6), and examples for different types of problem containing these terms are listed in Table 1. **f**, Sources of discretization in all the stochastic processes (either conventional or neuromorphic sources) impacts the accuracy and convergence of the expectation solution for the PIDE. The first row details the sampling order of convergence; the second row is the order of convergence for the Euler–Maruyama discretization method; the third row is the best-case scenario estimate for error accrued due to discretization space; the fourth and fifth rows merely indicate that some problems could have additional error due to the enforcement of a finite state space or due to reduced precision on neuromorphic platforms. Methods provides additional details. The colour bars represent the count of walkers at each location.

Extended Data Fig. 1 provides the additional circuit details. In operation, an initial count of walkers is set at the appropriate starting node (either through input spikes or an initial voltage condition). Once the supervisor circuit initiates the model, the spikes’ propagation through the mesh directly reflects the movement of random.

---

| **Table 1** |
| --- |
| **Problem** | **Approximation** | **Visualization** | **Error/convergence** |
| Sampling stochastic process | \( u(t, x) \approx \frac{1}{N^2} \sum_{i,j=1}^N g(t, X^i_0; X^j_0 = x) \) | | \( \frac{1}{\sqrt{Nt}} \) |
| Time discretization of random walks | \( u(t, x) \approx \frac{1}{N} \sum_{j=1}^{N} g(t, X^j_0; X^j_0 = x) \) | | \( \sqrt{t/N} \) |
| Spatial discretization of random walks | \( u(t, x) \approx \frac{1}{N^2} \sum_{i,j=1}^N g(t, X^i_0; X^j_0 = x) \) | | \( \frac{1}{N^{2/3}} \) |
| Finite range of random walks | \( u(t, x) \approx \frac{1}{N} \sum_{j=1}^{N} g(t, X^j_0; X^j_0 = x) \) | | Varies |
| Limited-precision probabilities | \( P \approx \frac{1}{256} \) | | Varies |

---

neural circuits. One circuit uses common leaky integrate-and-fire (IF) neurons (Fig. 1a) to count the number of incoming spikes; the other stochastically distributes spikes to output nodes (Fig. 1b). These mesh-point nodes are assembled into a graph whose edges represent the transition probabilities from one state to another (Fig. 1c).
Fig. 3 | Particle transport simulations on neuromorphic hardware. a, Top: non-spatial Boltzmann transition/absorption model. Bottom: the corresponding DTMC approximation for the underlying SDE. b, Evolution of particles through Boltzmann transitions on TrueNorth. The pink colour represents a higher density of walkers and the blue colour represents a lower density for the case where 1,000 walkers start in the +1 (top) or -1 (bottom) state and equilibrate due to Boltzmann transitions. The colour bar refers to count of walkers in each node. c, PIDE solution calculated through TrueNorth spike data starting from 1,000 random walkers along each direction. d, PIDE solution calculated through TrueNorth spike data starting from 10,000 random walkers along each direction. e, Spatial particle transport model. The particles in the physical problem exist in three dimensions and move at a constant speed. When experiencing a random scattering event (red dot), they change the angle, or direction, of their travel. When projected onto one dimension, this change in angle represents a change in the particle’s relative velocity in one dimension. f, MATLAB Euler–Maruyama using 1,000 random walkers per location at different time resolutions. g, MATLAB simulation of the approximate solution from the DTMC implementation of a spatial particle model with one million walkers per starting location. h, Intel Loihi simulation of the approximate solution from DTMC implementation with 6,250 walkers per starting location. The colour bar in h applies to panels f–h and represents the simulated particle fluence. i, Absolute error between the Loihi and MATLAB simulations. The colour bar refers to the percent error j. Average percent error between Loihi and numerical simulation as a function of increasing random walkers per starting location.

walks through the corresponding state space. Stated differently, the neuromorphic computing hardware implements both stochastic and deterministic components of the stochastic process.

Importantly, this neural algorithm can be generally configured to represent any time-homogeneous discrete-time Markov chain (DTMC) by configuring the shape of this graph and setting the output probabilities within each node to represent the problem description (Extended Data Fig. 2 shows an example). For instance, a nearest-neighbour 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 random walks, including those with non-local and jump diffusion as well as walker absorption and creation, can be readily implemented with location-dependent transition probabilities in this framework, allowing the algorithm to realize the processing-in-memory advantages of neuromorphic computing.

We first performed scaling studies to assess the computational costs inherent in simulating random walks on two neuromorphic computing platforms, namely IBM TrueNorth and Intel Loihi, relative to a commodity server-class Intel Xeon E5-2662 central processing unit.
Table 1 | Examples of applications involving a PIDE in the form of equation (4)

| Non-zero terms in equation (4) | Example application |
|-------------------------------|---------------------|
| $a$, $b$, $c$, $f$ | Stock option pricing $^{a3}$ |
| $\lambda$, $b$, $c$, $f$, $h$ | Boltzmann flux density (Supplementary Note 3) |
| Reduced problem (Fig. 3a–d) | |
| $a$, $c$ | Heat equation with dissipation (Fig. 4c) |
| $a$, $f$ | Electrostatic scalar potential, heat transport or simple beam bending $^{c}$ |
| $\lambda$, $b$, $c$, $f$, $h$ | Particle fluence (Supplementary Note 3) |
| Reduced problem (Fig. 3e–i) | |

This table is not exhaustive and only includes a sample of possible applications. In this Article, we utilize a random walk method to solve two heat transport problems and a reduced problem for both Boltzmann angular flux density problem and angular fluence problem.

Neuromorphic random walks apply to a broad class of PIDEs

Random walk solutions to PIDEs are used in a variety of fields, including computer science, physics, medicine and operations research $^{d}$. Typically, these are approached in a Monte Carlo fashion, sampling some underlying space to perform high-dimensional integration. The approach we use is also Monte Carlo in nature. We use stochastic calculus to identify a random process underlying a particular class of PIDEs. We then sample this process to estimate an expected value.

Consider an initial-value PIDE of the form

$$
\begin{align*}
\frac{\partial}{\partial t} u(t, x) &= \frac{1}{2} \sum_{i,j} \left( a_{ij} \right) \frac{\partial^2}{\partial x_i \partial x_j} u(t, x) + \sum_{i} b_i(t, x) \frac{\partial}{\partial x_i} u(t, x) \\
+ \lambda(t, x) \int_{\mathbb{Q}} u(t, x + h(t, x, q)) - u(t, x) \, \phi_Q(q; t, x) \, dq \\
+ c(t, x) u(t, x) + f(t, x), \quad x \in \mathbb{R}^d, \quad t \in [0, \infty)
\end{align*}
$$

(1)

where $f$ is the diffusion coefficient function $h$ describes the drift vector $\lambda$ is the divergence coefficient $c$ is a forcing term $\phi_Q$ is a probability density function $\mathbb{Q}$ is a probability space $T$ is a terminal time $e$ is a probability distribution.

Under some mild conditions $^{c3}$, if the PIDE has a solution, then that solution can be written as

$$
\begin{align*}
\mathbb{E} \left[ g(X(t)) \exp \left( \int_0^t c(s, X(s)) \, ds \right) \right] \\
+ \int_0^t \mathbb{E} \left[ g(X(t)) \exp \left( \int_s^t c(\ell, X(\ell)) \, d\ell \right) \right] \, ds \big| X(0) = x
\end{align*}
$$

(2)

This notation is an expected value, $\mathbb{E}$ conditioned on the starting location of the random process, $X(0)$.

The process $X(t)$ is a drift-diffusion process with a jump term, written as an SDE. Function $b$ defines the drift, $a$ gives the diffusion strength and $W(t)$ represents Brownian motion with respect to the underlying space (in this case, $\mathbb{R}^d$). The term $P(t; Q, X(t))$ is a Poisson process with the parameter given by $-\int_0^t \lambda(s, X(s)) \, ds$ and $\phi_Q$ describes the increment awarded whenever the Poisson process fires. The jump-amplitude mark random variable is $Q$, with probability density $\Phi_Q$. This stochastic process is visualized in Fig. 2a–c for constant values of $b$, $a$ and $h$. The jump value need not be constant and can even be random (Fig. 2d). Figure 2e,f showcases when the jump value is uniformly drawn over $\{-3, -2, \ldots, 2, 3\}$. Although $c$ does not appear in equation (3), it can often be interpreted as an absorption term (Fig. 2e). Supplementary Note 2 provides more information.

In this Article, we also consider a steady-state example where $\lambda = 0$ and $t$ does not appear. Taking equation (1) as a steady-state boundary value problem with $c = 0$ and boundary condition $u(x) = \psi(x)$, where $x$ is on the boundary of some domain $D$, then the solution may be written as

$$
\mathbb{E} \left[ \psi(X(T_x)) + \int_0^{T_x} f(X(s)) \, ds \right] \big| X(0) = x
$$

(4)

if a solution exists. The stochastic process $X(t)$ is the same as given by equation (3), but with $t$ omitted as an argument in the functions $a$, $b$ and $h$. The term $T_x$ is the time for which the random process $X(t)$ starting at $X(0) = x$ exits domain $D$.

Both equations (2) and (4) are modifications of classic results, reformulated to be applicable to forward time or jump terms. Supplementary Note 2 provides more information. For our neuromorphic implementation, we recognized that DTMC approximations to the SDEs make the probabilistic sampling of paths viable on the neuromorphic computing diffusion algorithm.
IBM’s TrueNorth and Intel’s Loihi pseudo-random number generators (PRNGs) are effectively limited to 8 bits. In a special best-case scenario (Supplementary Note 1), additional error could arise from hardware-specific limitations. For instance, in a neural algorithm that we use, the error is analogous to 7-bit probabilities in MATLAB. The DTMC approximation of a stochastic process introduces error. The implication of these errors will differ considerably across applications in practice.

Both conventional and neuromorphic simulations are impacted by 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 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 represent state variables raises the cost of reducing the meshing error. The implication of these errors will differ considerably across applications in practice.

The concept of sampling using neuromorphic computing has been previously explored for distributions. These efforts run a chain until stationary distribution is achieved and then sampling from this stationary distribution. In contrast, our approach embeds a distribution directly into a graph and samples throughout time. We do not make claims that our method of implementing a Markov chain on neuromorphic hardware is better than other methods or even that the choices we make in approximating an SDE with a DTMC are the best; rather, we merely point out that these tools alone are enough to allow for solving a large class of PIDEs using neuromorphic computing.
Approximating the solution to a PIDE using our neuromorphic computing algorithm follows these steps.

1. Determine the underlying SDE from the PIDE (equation (3))
2. Approximate the SDE through a DTMC (Methods)
3. Implement the DTMC on a neuromorphic platform (Methods)
4. Gather samples from the DTMC
5. Average the samples to produce the approximate solution (Methods)

Steps 1 and 2 above describe the process of moving from equation (3) to a DTMC. Although moving from the PIDE to the SDE is as easy as identifying the appropriate components common to equations (1) and (3), the discretization of the SDE to a DTMC requires some care. If a discrete, finite state space was not necessary, we could simply sample the SDE according to an appropriate scheme\(^{34,35}\). We give a prescriptive treatment on constructing the DTMC in Methods and provide a sample DTMC diagram in Extended Data Fig. 2. The basic steps, however, are to select an appropriate discretization of the SDE range to use as the state space and then integrate the distribution of the time-discretized SDE between appropriate bounds to determine transitions, similar to older methods for sampling jump diffusions\(^{36}\).

Examples of neuromorphic random walks

To demonstrate the ability of neuromorphic hardware to implement the DTMCs required for solving these PDEs, 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 Figs. 3 and 4. We provide more salient points of these examples in the next two subsections and the remaining details are available in Supplementary Note 3.

Neuromorphic hardware can simulate particle transport

The first of our two particle transport problems is an initial-value time-dependent problem detailing the angular flux density of a hypothetical particle (Fig. 3a). Consider a particle that exists in one of the two states: +1 or –1. We will call these states as the particle’s direction. The particle may ‘scatter’, uniformly choosing a new direction from ±1 according to a Poisson process with rate \(\sigma\). The particle may also be absorbed according to a second Poisson process with rate \(\tau\). The two rates correspond to \(\lambda\) and \(c\), respectively, in equation (1). The change in direction, relative to the current direction, after scattering is \(h\). All the remaining terms, namely, \(a\), \(b\) and \(f\), are zero. Given an initial condition \(g\), the angular flux density, \(\Phi(t,\Omega)\), of a population of these particles obeys the Boltzmann transport equation (Supplementary Note 3).

The underlying stochastic process (equation (3)) for this problem updates the direction value according to a Poisson process. Supplementary Note 3 provides more information. We deployed the DTMC approximation of this process on TrueNorth. The obtained neural approximation well approximates the analytical solution using only 1,000 walkers per starting condition (Fig. 3c). Moving to 10,000 random walks per starting position (Fig. 3d), we see a notable improvement in the approximation.

This simplified example of particle transport has broad implications. Directly, if we can well approximate the solution for this reduced problem, then it will be possible to approximate more complicated transport problems where a simulation is preferred to analytical methods. To that end, we have examined a second particle-transport-inspired example for which no analytical solution is readily available.

For this second example, we considered a particle that may travel in any direction in \([-1, 1]\). The particle scatters with a Poisson rate \(\sigma_s\), but does not experience absorption. The particle travels in space at speed \(v\) in direction \(\Omega\) (Fig. 3e). Given a source term \(S(x,\Omega)\), angular fluence \(\mathcal{U}(x,\Omega)\) obeys the steady-state Boltzmann transport equation. This is a steady-state version of equation (1), \(\lambda = \sigma_s\), \(f = vS(x,\Omega)\) and \(b = v\Omega\); \(h\) is the change in direction after scattering. Both \(a\) and \(c\) are zero; with absorbing boundary conditions in space, the probabilistic solution then takes the form of equation (4).

The underlying stochastic process is a vector process. The first component, \(X(t)\), describes the position of a random particle. It travels with velocity \(-\mathcal{U}(t)\), where \(Y(t)\) is the second component describing the direction. The direction remains constant until the scattering Poisson process fires, updating the direction. This process appears to move against the physical particle. Additional details on the DTMC and parameters are provided in Supplementary Note 3.

We completed a DTMC simulation with one million walkers per location in MATLAB on a \(30\times30\) mesh to use as a baseline comparison (Fig. 3g). Like TrueNorth, implementing this simulation on Loihi replicated the numerical examples (Fig. 3h) with a low overall error (Fig. 3i,j). 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.

The Euler–Maruyama method provides the basis for our DTMC approximation (Methods). This method improves with a decreasing time-step size. Although exceedingly fine time steps are not currently possible, we can simulate this method to demonstrate the improved quality of simulation expected through the better scaling of neuromorphic hardware. On a \(60\times60\) mesh (locations \(\times\) directions), we completed the simulations of the Euler–Maruyama method using 1,000 walkers per location in MATLAB (Supplementary Note 3), varying the time discretization to demonstrate this expectation.

Non-Euclidean neuromorphic random walks

The particle examples are straightforward demonstrations of random walks with non-local jumps on a simple domain. For solving PDEs over larger or more complex geometries, the neuromorphic random walk method can be utilized by carefully defining a mesh and calculating the transition probabilities. We demonstrated neuromorphic random walks over non-Euclidean domains by solving two heat equations involving spheres. Although the shapes that we consider are by no means ‘complex’ (Extended Data Fig. 4), we merely showcase that the method is mostly agnostic to the domain.

The first non-Euclidean problem that we considered is a basic heat equation on the unit sphere without dissipation. We let \(\mathbb{S}^2\) represent the unit sphere and assume \(\Phi(t,x)\) to be the positive scalar \(\Phi\) for all \(t \in [0,\infty)\) and \(x \in \mathbb{S}^2\). We set the remaining coefficients in equation (4) to zero. Paired with the initial condition \(\Phi(x)\) (Fig. 4a), the probabilistic solution to the heat equation on the sphere is given by equation (2), where the stochastic process is merely Brownian motion on the surface of the sphere with diffusivity \(\sqrt{2\alpha}\).

We simulate the underlying process on Intel’s Loihi platform. Starting with 3,000 random walks on each position yields the approximate solution shown in Fig. 4a for a collection of time points. The nature of the Monte Carlo process and DTMC approximation imply that we would see better agreement to the analytical solution with a finer mesh and a greater number of walkers per starting location. Although our neuromorphic computing implementation is qualitatively accurate, it transiently shows more error relative to the analytical solution compared with a numerical solution (Fig. 4b). We artificially lowered the precision of transition probabilities in the numerical solution, observing that the Loihi solution showed an error profile equivalent to representing transition probabilities at 7 bits, which is roughly consistent with the Loihi transitions governed by an 8-bit PRNG (Supplementary Note 3).

This example provides compelling evidence that complex geometries in which analytical methods are less tractable remain an
opportunity for neuromorphic computing to have a definite impact. As a final test case for our neuromorphic random walks on a more interesting geometry, we considered an initial-value heat flow problem on a barbell with dissipation. Again, we set \( a(t,x) = \kappa \) (some positive scalar) and take \( c(t,x) = \alpha \) (another positive scalar) for dissipation. All the other coefficients in equation (1) are assumed to be zero. Letting \( g(x) \) to be an initial condition, the probabilistic solution is given by equation (2) and the underlying stochastic process is Brownian motion on the barbell.

Our discretization scheme for this DTMC required 748 mesh points (Supplementary Note 3 provides details on the mesh construction and DTMC). Due to the restriction on the mesh size relative to the currently limited neuromorphic chip sizes available to us, we deployed this example on a spiking net simulator. The simulation results for various time points are shown in Fig. 4c. The temperature equilibration of the left (Fig. 4d) and right (Fig. 4e) sides of the barbell proceed as one would expect from thermodynamics.

Conclusions

We have shown that spiking neuromorphic hardware technology is suitable for implementing a scalable energy-efficient approach to resolve an important set of numerical computing problems. In particular, we show that our neural random walk algorithm scales comparably to a parallel CPU approach, showing a substantial energy advantage. We have also shown 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, such as 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.

We have focused our demonstrations on large-scale digital spiking platforms—Loihi and TrueNorth—because they have the requisite neural scales for our algorithms. We expect that other large-scale neuromorphic computing platforms, like the ARM-based SpiNNaker and analogue complementary metal–oxide–semiconductor BrainScaleS, would scale similarly. In particular, the high clock speed of BrainScaleS-2 arising from its mixed-signal digital and analogue processing (up to 100 MHz (ref. 38)) would likely put it on a different part of the Pareto frontier of speed and energy costs (Fig. 1). There is also considerable interest in neuromorphic approaches leveraging emerging non-complementary metal–oxide–semiconductor analogue devices, including those that leverage memristive devices for synapses and neurons, although we would have to consider the precision implications of analogue devices alongside other approximation considerations (Fig. 2b).

Our results provide insights into future considerations for neuromorphic computing hardware design. There are several features of neuromorphic computing hardware that our algorithm heavily leverages. Foremost, our algorithm requires a careful consideration of how random numbers are generated in parallel. Most current neuromorphic computing approaches leverage conventional PRNGs that are shared across many neurons; however, this could change considerably as physically noisy analogue components are explored for use as neurons and synapses. These results, thus, provide a mathematically interpretable use case for taking advantage of device stochasticity.

Additionally, the approach described here would benefit from the event-driven communication of labelled spikes, which would make path-dependent Monte Carlo strategies more efficient. Likewise, the neurons used in our algorithm are far simpler than biological neurons and an architecture specialized for our random walk approach could achieve considerably higher densities than existing neuromorphic computing platforms by using simpler neurons.

Neuromorphic computing also potentially offers capabilities that our random walk algorithm does not yet leverage. Most notably, the learning capabilities on platforms such as BrainScaleS and Loihi could be used to directly account for changes within the modelled domain or as a tool to infer unknown parameters for a stochastic process that best explain the experimental observations. Similarly, although our algorithm leverages spike timing and delays to gain its advantage, our current algorithm requires a level of spike-timing precision on TrueNorth and Loihi that is far greater than the brain appears to use, suggesting that further improvements could be obtained by using brain-inspired timing strategies that permit lower temporal precision. This high temporal precision is specific to our neuromorphic implementation and is not fundamental to our DTMC neural algorithm: the only critical timing for DTMC is that all the walkers are on the same model time steps, and even that requirement can be bypassed for steady-state problems.

Our approach avoids the approximation pitfalls associated with many AI algorithms, wherein the implications of numerical precision and interpretability are still an open question. Although understanding and accounting for these approximation errors will be critical for any application, the graph-based approach taken here provides several well-understood design choices to appropriately tailor the algorithm and hardware solution given the precision concerns. Solving PIDEs at scale often requires high-performance computing systems today, and we expect this neural approach to similarly scale beyond single-chip systems. Early indications are that neuromorphic computing scales effectively to multiple chips and boards due to its lightweight spiking communication. We expect that neuromorphic hardware will eventually exist primarily in heterogeneous system architectures alongside CPUs, GPUs and other accelerators.

Despite its intrinsic advantages of low power and parallel computing, as well as its implicit promises of new brain-derived capabilities, the ultimate use case of neuromorphic computing remains an open question. It is unclear whether the best long-term opportunity for neuromorphic systems will be to become a GPU-like coprocessor that offers improved capabilities for a range of tasks, or if its biggest impact will be in specialized low-power sensors and accelerators. We believe our results support the potential of more flexible neuromorphic systems, as the algorithms for solving PIDEs 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. 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 can deliver concrete energy advantages and is capable of effectively scaling in terms of processing time and overall efficiency.

Methods

We subdivide our Methods section into two main components: neuromorphic hardware methods and mathematical methods. In the former, we provide details on the random walk circuit and model implementation. In the latter, we describe one method for approximating an SDE with a finite state space, namely, the DTMC. We also showcase how to average the random walks to approximate the solution to a PIDE. Supplementary Note 1 provides additional details about the neuromorphic methods. Supplementary Note 2 provides additional details about the mathematics. Supplementary Note 3 provides additional details about the examples.

Neuromorphic hardware methods. General. The neurons used in both Loihi and TrueNorth are either IF neurons or TG neurons. In both cases, the hardware neurons integrate all the 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 experience full decay for 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 elsewhere. Each mesh point in the simulation consisted of two counting circuits (one to buffer the inputs
and the other to count down the outputs) and a probabilistic fan-out circuit. The network also utilizes a population of supervisor neurons to control the timing and synchrony of the walkers through the circuit. For cases where walkers are synchronized with the time step, a separate buffer circuit is required to accumulate all the 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, although with some minor differences that are noted.

**Buffer and counter circuits.** In the random walk algorithm, the buffer circuit and walker-counting circuit are almost identical, with the only difference being the inputs and outputs. The counting circuits on Loihi are structured as shown in Extended Data Fig. 1a. 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 stop the count neuron if the 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’s 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 true 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.

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 time step, if there are k walkers at a location, the generator neuron will fire (k + 1) times (which is subsequently corrected). During the first half of the simulation time step, the buffer generator neuron transfers the count from the buffer to the counter circuit; during the second half of the simulation time step, the counter generator neuron transfers the count to the probabilistic fan-out circuit (described below), which can then distribute 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 synapses.

**Probability circuit on TrueNorth and 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 PRNGs available to each individual neuron to select only one of the mesh node’s outputs at the appropriate Markov transition probability.

Although there are several probable implementations of a circuit to send a spike to one of the N outputs with probability p_in (i = 1 … 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, which potentially could be stochastic and additive scaling. On TrueNorth, we take advantage of its neurons’ stochastic leak capabilities, which similarly provides an 8-bit random value. In both cases, the approach is to generate a neuron, which on receiving an input, will fire at probability p.

For both platforms, we consider a conceptual binary probability tree for which the leaf has probabilities of firing that reflect the Markov transition probabilities of the mesh node (Extended Data Fig. 1b). This tree will have a depth of \(O(\log[N])\). On Loihi, we developed a procedure to collapse this probability tree into a circuit that requires (N − 1) probability neurons in a single hidden layer with precise excitation and inhibition onto a set of output neurons (Extended Data Fig. 1c). Supplementary Note 1 provides the algorithm for this procedure. On TrueNorth, we directly implemented this binary probability tree in a circuit (Extended Data Fig. 1d).

**Implementing models on TrueNorth and Loihi.** 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. Particularly for TrueNorth, Extended Data Fig. 1e shows a connectivity diagram, which provides the details of neuron connectivity for implementing the mesh node in TrueNorth; this is the TrueNorth model representation of the circuit shown in Extended Data Fig. 1d. Importantly, different mesh points can have distinct numbers of outputs, although more outputs will directly increase the number of neurons required. Further, there is no real restriction on the types of graph and connectivity, although 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.

Since neuromorphic cores on TrueNorth and Loihi have a limited number of neurons (256 on TrueNorth and up to 1,024 on Loihi), most models are distributed over multiple neuromorphic cores. The method used to implement stochastic neurons on TrueNorth allows both stochastic and deterministic neurons to exist on the same core, whereas our implementation on Loihi required different cores for the deterministic and stochastic neurons (Extended Data Fig. If). As both hardware platforms have many cores per chip, in both cases, we manually compiled the different neuron types of mesh points onto specific neuromorphic cores on each chip. As the community’s tools for neuromorphic graph embeddings are improved, we expect that the implementations we used could be optimized in the future.

For most of the examples in the main text, the specific Loihi and TrueNorth neuromorphic models remain unchanged, with the only variable input being the transition matrix used to define the connections between the mesh nodes and weights onto the probability neurons in the mesh points. Complete details of generating Markov transition matrices for the examples discussed in the main text used on Loihi and TrueNorth are given in Supplementary Note 3.

**Scaling studies: execution and statistics.** We performed a number of scaling experiments on TrueNorth and Loihi. The base experiment is a random walk simulation on a 21 x 21 node torus mesh, where each node has four incoming and four outgoing connections (corresponding to a walker on the surface moving up, down, left or right). The transition probabilities of all the four directions are 0.25. All the walkers begin the simulation at the centre of the mesh, and the simulations are run for at least 100,000 model time steps (the unit of time required for all the walkers to have moved one position).

Initially, the number of walkers is increased from 1,000 to 32,000 (Fig. 1e, red line) on a single mesh. To compare with weak scaling across multiple conventional cores, on TrueNorth, we increased the number of walkers through parallelization, which involved making copies of the underlying mesh, giving 1,000 random walkers each. The mesh copies increase to keep the total walkers in the simulation consistent with the static mesh runs.

For each experiment, we performed ten replicate trials to estimate the mean and standard deviation. Further descriptions of the scaling studies are given in Supplementary Note 1 and Extended Data Fig. 3.

**Computer simulations.** For validation of our algorithm and simulations of the barbell experiment, we performed simple direct simulations of the neural algorithm dynamics. This is a simple, direct Python discrete-time simulation of the neural algorithm, and not a formal simulator like NEST or a model description language like PyNN. For this, we used the reference simulator described elsewhere.[2]

**Mathematical methods.** The success of approximating solutions to PIDEs hinges on the ability to implement a random walk approximation of the underlying stochastic process. There are many ways to construct such an approximation; we highlight our choices and point out where variations can occur.

**Construction.** We construct a DTMC approximation for a one-dimensional SDE. The method described can be naturally extended to higher dimensions. Consider the following SDE:

\[ dX(t) = (b(t, X(t)) dt + a(t, X(t)) dW(t)) + \Phi(t; Q, X(t)), \]

(5)

Here \( b \) represents the drift of the process \( X(t) \) and \( a \) represents the diffusion. A non-local diffusion, or jump term, is governed by process \( P \) with reward \( h \). The term \( Q \) is the jump-amplitude mark random variable with probability density function given by \( \Phi(t; q, x) \).

For discussion, we assume that \( X(t) \) is \( R \) and \( a, b \) and \( h \) are such that \( X(t) \) can assume any value in \( R \) with non-zero probability. The Markov chain approximation constructed must take values from a finite set. We will need to divide the real line into a finite number of intervals to represent our state space and determine the probability of transition between these intervals using equation (5).

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 process \( X(t) \) can transition to any interval on the real line. If the number of hardware-allowable transitions is limited, care must be taken to conserve probability. That is, the sum of allowable transitions must be equal to 1.

We will carefully explore the nuances of approximation, 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 the neighbours for each state in the Markov chain
3. Calculate the probability for each transition in the chain
4. Restrict to a finite state space

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

**Countable state space for the Markov chain.** To begin our approximation for equation (5), we must separate 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
Neighbours for each state in the Markov chain. As previously discussed, given a starting location, 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 interval in state space would be a neighbour to all the other states. That is, the graph representing the possible transitions between the state spaces would be a complete graph.

However, neuromorphic hardware and practical limitations (that is, fan-in/ fan-out considerations) suggest that we cannot allow each state to transition to all the others. In the context of neuromorphic hardware, we must decide how many fan-out we want. In the examples in the main 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 then back to itself (Extended Data Fig. 2, arrows).

Calculation of transition probabilities. The first two steps in our construction were independent choices: a choice of \( \Delta t \) and a choice of neighbouring states. To calculate the transition probabilities, a dependent choice is made. Namely, we must select an appropriate time discretization size \( \Delta t \) for the DTMC.

Once equation (5) can transition to any interval on the real line given a starting position, we must choose \( \Delta t \) to be small enough so that the probability of transitioning to intervals outside of the defined neighbours is smaller than some threshold probability, which is set to 0.05. However, if there is a Poisson process, that is, if \( \Delta t \), then in the SDE, we then also want to choose \( \Delta t \) to be small enough so that we can reasonably sure that at most one Poisson event can occur in any time window. Again, ensure \( \Delta t \) is selected so that the probability of more than one Poisson event occurring in any time window is less than 0.05.

With \( \Delta t \), we can now calculate the transition probabilities. Suppose that \( x_i \) and \( x_j \) are the nodes and that \( x_i \) is a neighbour of \( x_j \). For simplicity, assume that \( h \) is deterministic. Then, using equation (5), the probability of \( x_i \) transitioning to \( x_j \) in the time interval \([t, t+\Delta t]\) is the sum of two probabilities: the probability that \( X(t) = x_i \) and \( X(t+\Delta t) \) is in the interval represented by \( x_j \) with and without a Poisson jump occurring.

We use the Euler–Maruyama discretization method to calculate these probabilities. The method is similar to the Euler method in that the value of \( X(t) \) is assumed to be constant over some interval \([t, t+\Delta t]\) and an increment is calculated from this assumption. For equation (5), given \( X(0) = x_i \) and assuming that \( h \) is deterministic, the Euler–Maruyama method gives

\[
X(t + \Delta t) \approx x_i + b(x, t) \Delta t + a(x, t) \cdot W(\Delta t) + h(x, t) \cdot \Delta t,
\]

where \( x_{f,i} \) is the indicator of whether the Poisson process \( P \) fired in the time window \([t, t+\Delta t]\) and \( W(\Delta t) \) is a normal random variable with mean 0 and variance \( \Delta t \). Put another way, if the Poisson process does not fire, then \( X(t+\Delta t) \) is a normal random variable with mean \( x_i + b(x, t)\Delta t \) and variance \( a^2(x, t)\Delta t \). If the Poisson process does fire, then \( X(t+\Delta t) \) is a normal random variable with mean \( x_i + b(x, t)\Delta t + h(x, t) \) and variance \( a^2(x, t)\Delta t \). Given these representations as normal random variables, one can calculate the probability of transition for each into the interval represented by \( x_j \). Then, the probability of transition from \( x_i \) to \( x_j \) as time increases from \( t \) to \( t+\Delta t \) is the sum of the former times the probability that no jump occurs plus the latter times the probability of a jump occurring.

If jump \( h \) is random, a similar sum can be made by factoring in the distribution of rewards. Our steady-state particle transport problem is an example of this concept (Supplementary Note 3).

The calculated transition probabilities form a transition tensor \( C(t) = (p_{ij}(t)) \) representing the Markov chain. For each fixed \( t \), we must conserve probability. In the Markov chain, this means that \( \sum_j p_{ij}(t) = 1 \). In this discussion example, we are only allowing transitions to the left, right and back to the same node. However, the true process \( X(t) \) may 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. Although \( \Delta t \) 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. One option is to normalize the rows; another is to add the missing amount to one of the transitions. Unless stated otherwise in our examples, we conserve this as follows. If, in our example, we were calculating the transition probability to the left node, we would calculate the probability of transitioning to the left node or beyond. A similar process is followed for the right node. Again, since \( \Delta t \) 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.

Further, \( C \) depends on time step \( t \), whenever \( a, b, \lambda, h \) or \( \Delta t \) depends on \( t \). When these functions do not depend on time, a static transition matrix is created. If time dependence exists and if the maximum desired time is known, the transition tensor \( C \) can collapse into a single matrix. For example, if there are ten possible state spaces and it is only desired to simulate for 100 time steps, then a hundred \( 10 \times 10 \) transition matrices for each time step would become a single transition matrix over a state space of size 1,000.

Restriction to a finite state space. There may be situations that arise where the method we have described yields a countably infinite state space; further, when considering hardware limitations, such as 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 equation (5) on the real line, we would need to select a finite subset of our states for transitioning. Due to the construction of the neighbours, we select a minimum interval and maximum interval, keeping all the states between them. Without loss of generality, we suppose that the states are ordered; the minimum state corresponds to \( x_0 \) and the maximum state corresponds to \( x_N \).

For each \( t \), we select the \( N \times N \) section of the transition matrix \( C(t) \) corresponding to our truncated state space. To conserve probability, for each state, we add the total probability of a transition to a state that is less than \( x_i \) to the probability of transitioning to \( x_i \). A similar process is followed for a transition to a state greater than \( x_i \). 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 \( x_i \) is some node in the state space, then we write \( X(t; k\Delta t) \) for the location of the \( i \)th random walk at time \( k\Delta t \) that started at position \( x_i \).

Utilizing sampled random walks to approximate PIDE solution. The process \( X(t) \) is used to calculate the following solution

\[
\begin{align*}
  u(t, x) &= \mathbb{E} \left[ g(X(t)) \exp \left( \int_t^s (s, X(s)) \frac{dW(s)}{\Delta t} \right) \right] \\
  &= \mathbb{E} \left[ \frac{u(s, X(s))}{\Delta t} \right],
\end{align*}
\]

We would like to evaluate \( u(t, x) \) via this expectation using the Monte Carlo method and the sampled random walks for each position \( x_i \) in the mesh and time point \( t \). Since our random walks occur over discrete time points and we only sample discrete locations, we do this via Riemann sum approximations for the integrals. If \( M \) total random walks were sampled that started on position \( x_i \), then

\[
\begin{align*}
  u(t, x_i) &\approx \frac{1}{M} \sum_{k=1}^M \left[ g(X_{i,k}(t)) \exp \left( \sum_{j=1}^{k-1} \left( \frac{t_{j+1} - t_j}{\Delta t} \right) \right) \right] \\
  &\quad + \sum_{k=1}^M \left[ f(X_{i,k}(t), X_{i,k}(t)) \right] \Delta t,
\end{align*}
\]

Source availability. Data source is provided with this paper. The computational scaling data generated and analysed in this study are included in the published article as Extended Data.

Code availability. The code that supports the findings of this study is available from the corresponding author upon reasonable request and concurrence with the US DOE and relevant hardware partners.

Received: 26 May 2021; Accepted: 16 December 2021; Published online: 14 February 2022

References

1. Davies, M. et al. Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro 38, 82–99 (2018).
2. Furber, S. B., Galluppi, F., Temple, S. & Plana, L. A. The SpiNNaker project. Proc. IEEE 102, 652–665 (2014).
3. Merolla, P. A. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science 345, 668–673 (2014).
4. Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. 41, 303–332 (1999).
5. von Neumann, J. The Computer and the Brain (Yale Univ. Press, 2000).
6. Roy, K., Jaiswal, A. & Panda, P. Towards spike-based machine intelligence with neuromorphic computing. Nature 575, 607–617 (2019).
7. Severa, W., Vineyard, C. M., Dellana, R., Verzi, S. J. & Aimone, J. B. Training deep neural networks for binary communication with the Whetstone method. Nat. Mach. Intell. 1, 865–94 (2019).
8. Shrestha, S. R. & Orchard, G. Slayer: spike layer error reassignment in time. In Proc. 32nd International Conference on Neural Information Processing Systems 1419–1428 (2018).
9. Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. Deep learning in spiking neural networks. Neural Netw. 114, 47–63 (2019).

10. Pfeiffer, M. & Pfetsch, T. Deep learning with spiking neurons: opportunities and challenges. Front. Neurosci. 12, 774 (2018).

11. Göltz, J. et al. Fast and energy-efficient neuromorphic deep learning with first-spoke times. Nat. Mach. Intell. 3, 823–835 (2021).

12. Aimone, J. B. Neural algorithms and computing beyond Moore's law. Commun. ACM 62, 110 (2019).

13. Lanyon, B. P. et al. Towards quantum chemistry on a quantum computer. Nat. Chem. 2, 106–111 (2010).

14. Feynman, R. P. Quantum mechanical computers. Found. Phys. 16, 507–531 (1986).

15. Harrow, A. W. & Montanaro, A. Quantum computational supremacy. Nature 549, 203–209 (2017).

16. Arute, F. et al. Quantum supremacy using a programmable superconducting processor. Nature 574, 505–510 (2019).

17. Parekh, O., Phillips, C. A., James, C. D. & Aimone, J. B. Constant-depth and subcubic-size threshold circuits for matrix multiplication. In Proc. 30th Symposium on Parallelism in Algorithms and Architectures 67–76 (ACM, 2018).

18. Wolpert, D. H. & Macready, W. G. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67–82 (1997).

19. Siu, K.-Y., Roychowdhury, V. & Kailath, T. Discrete Neural Computation: A Theoretical Foundation (Prentice-Hall, 1995).

20. Aimone, J. B. et al. Non-neural network applications for spiking neuromorphic hardware. In 3rd International Workshop on Post-Moore’s Era Supercomputing (PMES, 2018).

21. Aimone, J. B. et al. Dynamic programming with spiking neural computing. In Proc. International Conference on Neuromorphic Systems 20 (ACM, 2019).

22. Mnii, Z. M. Graph partitioning as quadratic unconstrained binary optimization (QUBO) on spiking neuromorphic hardware. In Proc. International Conference on Neuromorphic Systems 4 (ACM, 2019).

23. Schuman, C. D. et al. Shortest path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation. In Proc. 7th Annual Neuro-inspired Computational Elements Workshop 3 (ACM, 2019).

24. Buesing, L., Bill, J., Nessler, B. & Maass, W. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. PLoS Comput. Biol. 7, e1002211 (2011).

25. Peccevi, D., Buesing, L. & Maass, W. Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. PLoS Comput. Biol. 7, e1002294 (2011).

26. Mendat, D. R., Chin, S., Furbur, S. & Andreou, A. G. Markov chain Monte Carlo inference on graphical models using event-based processing on the SpiNNaker neuromorphic architecture. In 2015 49th Annual Conference on Information Sciences and Systems (CISS) 1–6 (IEEE, 2015).

27. Fonseca Guerra, G. A. & Farber, S. B. Using stochastic spiking neural networks on SpiNNaker to solve constraint satisfaction problems. Front. Neurosci. 11, 714 (2017).

28. Hamilton, S. P., Slattery, S. R. & Evans, T. M. Multigroup Monte Carlo on GPUs: comparison of history-and event-based algorithms. Ann. Nucl. Energy 113, 506–518 (2018).

29. Hamilton, S. P. & Evans, T. M. Continuous-energy Monte Carlo neutron transport on GPUs in the shift code. Ann. Nucl. Energy 128, 236–247 (2019).

30. Severa, W., Lehoucq, R., Parekh, O. & Aimone, J. B. Spiking neural networks in hardware. Preprint at https://arxiv.org/abs/1705.06963 (2017).

31. Smith, J. D. et al. Solving a steady-state PDE using spiking networks and neuromorphic hardware. In Proc. International Conference on Neuromorphic Systems 27 (ACM, 2020).

32. Masuda, N., Porter, M. A. & Lambiotte, R. Random walks and diffusion on networks. Phys. Rep. 716, 1–58 (2017).

33. Hansson, F. B. Applied Stochastic Processes and Control for Jump-Diffusions: Modeling, Analysis and Computation (SIAM, 2007).

34. Kloeden, P. E. & Platen, E. in Numerical Solution of Stochastic Differential Equations (eds Kloeden, P. E. & Platen, E.) 103–160 (Springer, 1992).

35. Burdzy, K. & Chen, Z.-Q. Discrete approximations to reflected Brownian motion. Ann. Probab. 36, 698–727 (2008).

36. Skorokhod, A. V. Studies in the Theory of Random Processes Vol. 7021 (Courier Dover Publications, 1982).

37. Schemmel, J. et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In Proc. 2010 IEEE International Symposium on Circuits and Systems (ISCAS) 1947–1950 (IEEE, 2010).

38. Billardelle, S. et al. Structural plasticity on an accelerated analog neuromorphic hardware system. Neural Netw. 133, 11–20 (2021).

39. Indiveri, G., Linares-Barranco, B., Lengenstein, R., Delaforge, G. & Prodromakis, T. Integration of nanoscale memristor synapses in neuromorphic computing architectures. Nanotechnology 24, 384010 (2013).

40. Pickett, M. D., Medeiros-Ribeiro, G. & Williams, R. S. A scalable neuromorphic architecture with built with Mott memristors. Nat. Mater. 12, 114–117 (2013).

41. Krichmar, J. L., Severa, W., Khan, S. M. & Olds, J. L. Making BREAD: biomimetic strategies for artificial intelligence now and in the future. Front. Neurosci. 13, 666 (2019).

42. Schuman, C. D. et al. A survey of neuromorphic computing and neural networks in hardware. Preprint at https://arxiv.org/abs/1705.06963 (2017).

43. Meier, K. Special report: can we copy the brain?—The brain as computer. IEEE Spectr. 54, 28–33 (2017).

44. Davies, M. Benchmarks for progress in neuromorphic computing. Nat. Mach. Intell. 1, 386–388 (2019).

45. Davies, M. et al. Advancing neuromorphic computing with Loihi: a survey of results and outlook. Proc. IEEE 5, 911–934 (2021).

46. Indiveri, G. & Douglas, R. Neuromorphic vision sensors. Science 288, 1189–1190 (2000).

47. Aimone, J. B., Severa, W. & Vineyard, C. M. Composing neural algorithms with Fugu. In Proc. International Conference on Neuromorphic Systems 3 (ACM, 2019).

48. Bossy, M. & Champagnat, N. in Encyclopedia of Quantitative Finance (eds. Rama, C.) 1142–1159 (John Wiley, 2010).

Acknowledgements
We thank S. Plimpton and A. Baczewski for reviewing an early version of the manuscript and A. Moody, S. Cardwell and C. Vineyard for managing access to the TrueNorth and Loihi platforms. We acknowledge financial support from Sandia National Laboratories’ Laboratory Directed Research and Development Program and the US Department of Energy (DOE) Advanced Simulation and Computing Program. Sandia National Laboratories is a multi-program laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International for the US DOE National Nuclear Security Administration under contract DE-NA0003525. This article describes objective technical results and analysis. Any subjective views or opinions that might be expressed do not necessarily represent the views of the US DOE or the US Government.

Author contributions
J.D.S., B.C.F. and R.B.L. derived the mathematical results. J.D.S. and B.C.F. designed the geometry experiments. J.D.S., W.S. and J.B.A. designed the paper. W.S. and J.B.A. designed the particle experiments. J.D.S., W.S. and J.B.A. designed the geometry experiments. Any subjective views or opinions that might be expressed do not necessarily represent the views of the US DOE or the US Government.

Competing interests
The authors declare no competing interests.

Additional information
Extended data is available for this paper at https://doi.org/10.1038/s41928-021-00705-7.

Supplementary information
The online version contains supplementary material available at https://doi.org/10.1038/s41928-021-00705-7.

Correspondence and requests for materials should be addressed to James B. Aimone.

Peer review information
Nature Electronics thanks the anonymous reviewers for their contribution to the peer review of this work.

Reprints and permissions information
is available at www.nature.com/reprints.

Publisher’s note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© The Author(s), under exclusive licence to Springer Nature Limited 2022.
Extended Data Fig. 1 | Neural Circuits for Random Walk Algorithm. (A) 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. (B) Illustration of computing probabilistic circuit with a decision tree to compute probabilities with example output probabilities in red. (C) 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). (D) 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. (E) 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 panel D. 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 spikes.
Extended Data Fig. 2 | Markov chain. Illustration on the creation of a Markov chain on the real line.
Extended Data Fig. 3 | Diffusion Random Walks Scaling Studies. (A) Walker updates per second for a 1,000 (dark green) and 32,000 (light green) basic diffusion simulation across conventional and neuromorphic platforms. (B) Comparison of Loihi and single-chip TrueNorth to a single-core CPU simulation on normalized time of a simple diffusion simulation as a function of increasing random walkers. All times normalized to the time it takes to complete a simulation with 1,000 walkers. (C) Comparison of multi-chip TrueNorth to multi-core CPU and GPU simulations. GPU generates threads for all walker scenarios; GPU Single Block allocates only 1,024 threads for all walkers. (D) TrueNorth Execution time reaches a limit as mesh counts increase. (E) TrueNorth Execution time scales linearly with walker count, again, but also demonstrates the sensitivity of the algorithm to bottlenecks caused by uneven transition probabilities. (F) TrueNorth Execution time is dramatically reduced once all walkers do not start on the same position. (G) Time required for NVIDIA Titan XP GPU to simulate diffusion on a torus for 100,000 time steps as a function of the number of walkers. A fixed 1,024 threads are allocated for each trial. (H) Time required for NVIDIA Titan XP GPU to simulate diffusion on a torus for 100,000 time steps as a function of the number of walkers. For this weak-scaling experiment, a block of 1024 is added for every 1,000 walkers. (I) Time required for NVIDIA Titan XP GPU to simulate diffusion on a torus for a single time step as a function of the number of walkers. For this weak-scaling experiment, a block of 1024 is added for every 1,000 walkers.
Extended Data Fig. 4 | Details on Particle Transport and Non-Euclidean Meshes. (A) To avoid issues with random walks not ending exactly on the mesh, $\Delta x$ can be expressed as a function of both $\Delta t$ and $\Delta \Omega$ (see Eq. SN3.14). The marker on this plot shows the selection for our simulations. (B) Average rounding distance in a single time step across all directions determined by the given value of $\Delta \Omega$. (C) The approximate solution to Eq. SN3.12 is calculated using $\Delta t = 0.01$, $\Delta x = 1/15$ and the given value of $\Delta \Omega$ utilizing 1 million walkers per starting location. The absolute value of the difference of this average value and the average value calculated when $\Delta \Omega = 1/15$ is presented. In both panels, the blue circle indicates the value of $\Delta \Omega$ used in the Loihi simulation. (D) Visualization of mesh structure for heat transport examples in the sphere. The center of each triangle represents a location in the mesh or an element of the state space. (E) Visualization of the mesh structure in the barbell. The center of each triangle or rectangle represents a location in the mesh.