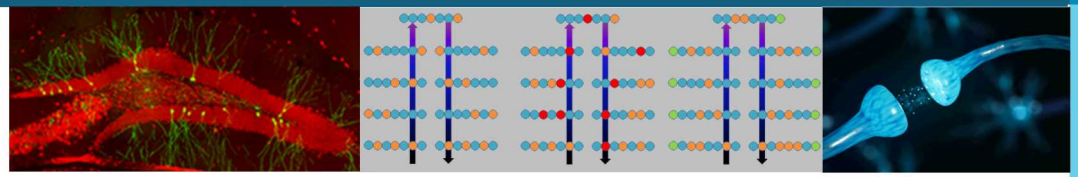
SAND2020-5297PE

A Neuromorphic Future for Classic Computing Tasks



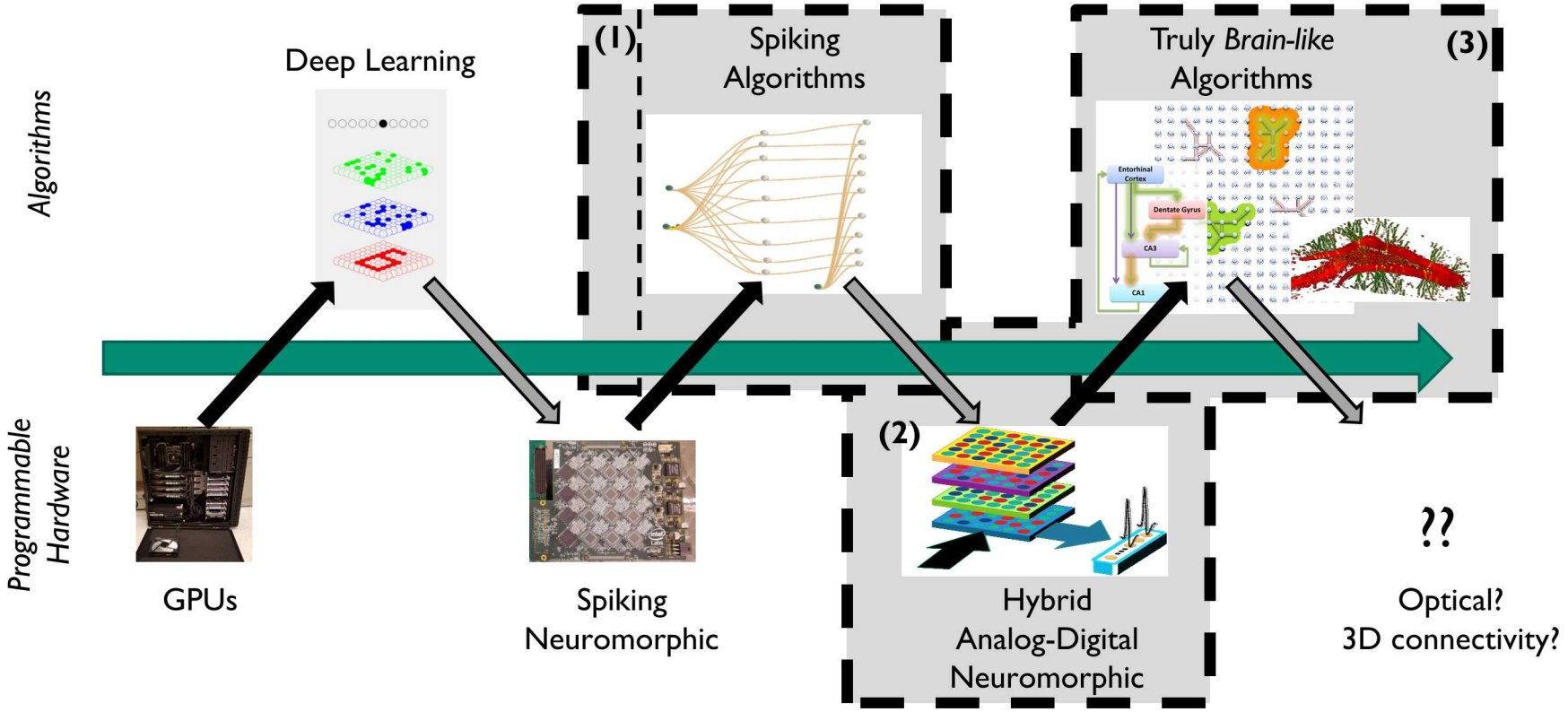
Presented by

Brad Aimone; jbaimon@sandia.gov

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.

SAND2020-1546 C

We see neuromorphic computing embarking on a fully co-design future



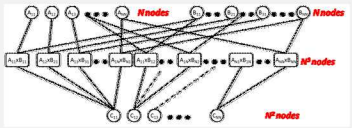
Positioning neuromorphic towards impact in many directions

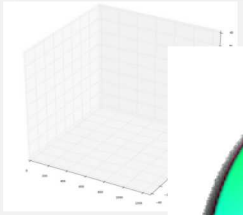
Scientific Computing

Well-understood requirements

Opportunity:

Novel neuromorphic algorithms





Examples:

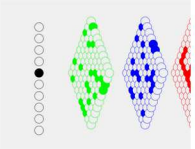
- Solving SDEs (Monte Carlo PDE solutions)
- Neural Graph Analytics

Machine Learning

Growing impact and need

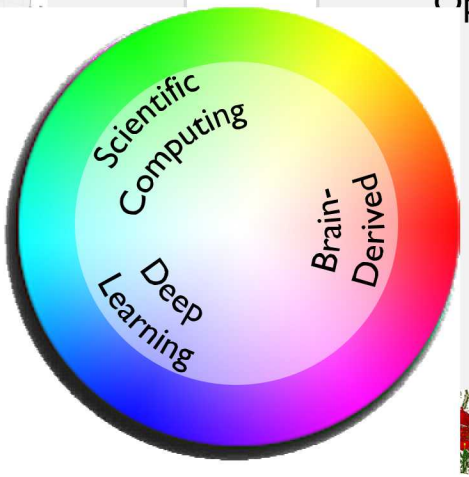
Opportunity:

Mapping to Neuromorphic



Example:

- Whetstone conversion of DL for spiking architectures



Brain-Derived AI

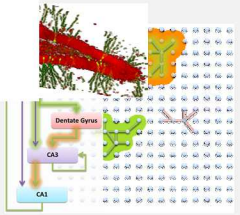
Achieve brain-like efficiency at advanced cognitive tasks, but path has proven elusive…

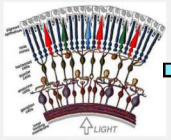
Opportunity:

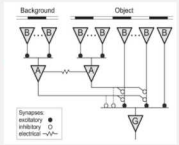
Develop novel algorithms that address critical DOE problems

Examples:

- Dragonfly model for interception
- Hippocampus model for context-dependent learning

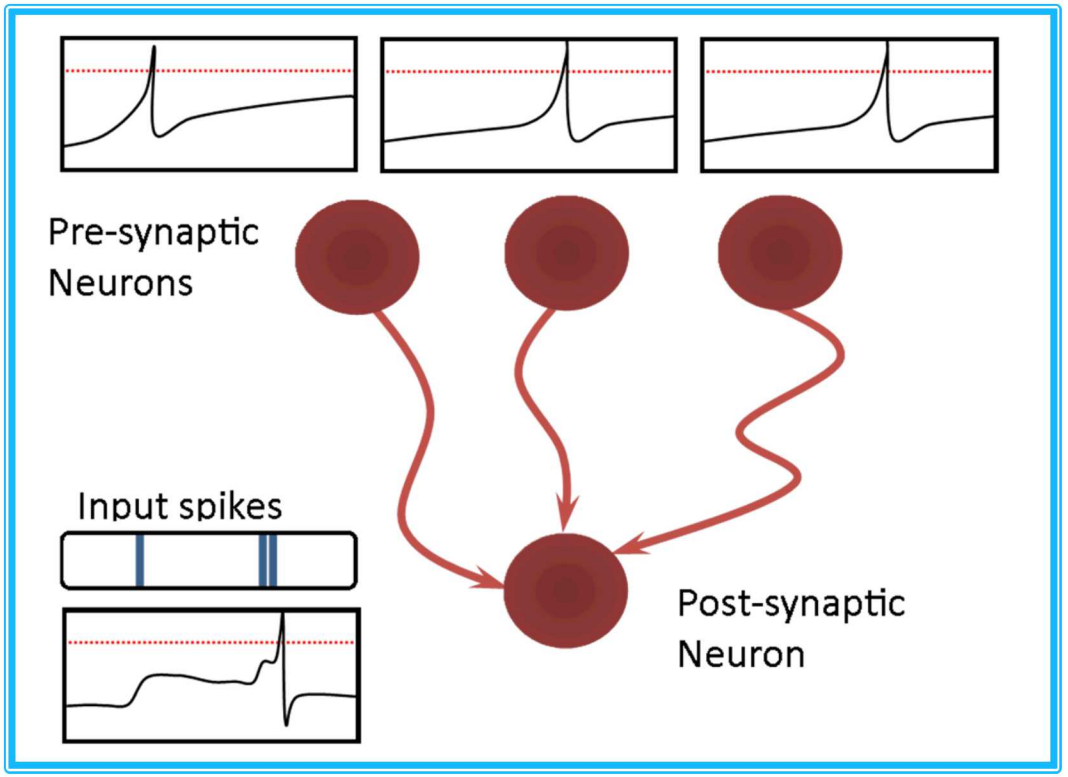






Spiking Neural Networks

- Subclass of Artificial Neural Network
- Neurons compute their own state independently, possibly asynchronously
- Each neuron integrates incoming information into a 'potential'
- If 'potential' reaches a predetermined threshold, the neuron alerts connected neurons
- Neuron communication is single-state signals (spikes)
- A time delay for spike propagation can be included
- Enables event-driven computation



Spiking Neural Networks – Neuron Dynamics

Generically, a discrete-time leaky-integrate-and-fire neuron is well-modeled by simulators and neuromorphic hardware.

For random draw η and weights $w_{i,j}$, delays $d_{i,j}$, initial voltages $V(0)$, probability of fire P_i, and initial action potentials $x(0)$ being algorithm dependent:

$$\widehat{V}_i(t+1) = V_i(t) + \sum_j w_{i,j} x_j\left(t - d_{i,j} + 1\right)$$

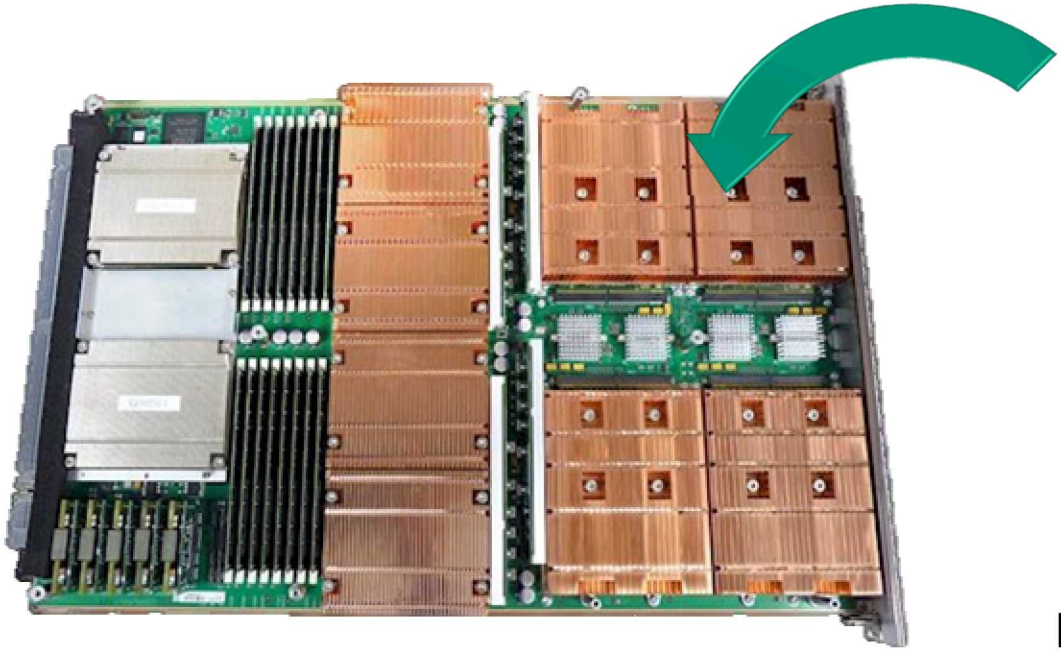
$$x_i(t+1) = \begin{cases} 1, & \hat{V}(t+1) > V_i^* \text{ and } \eta_{i,j} < P_i \\ 0, & \text{otherwise} \end{cases}$$

$$V_i(t+1) = \begin{cases} \tau_i V_i(t), & x_i(t+1) = 0 \\ 0, & x_i(t+1) = 1 \end{cases}$$

Each neuron processes these functions at **every time step** in **perfect parallel**.

If you can take your algorithm and formulate it
as a network of these independent processes, it can run on neuromorphic.

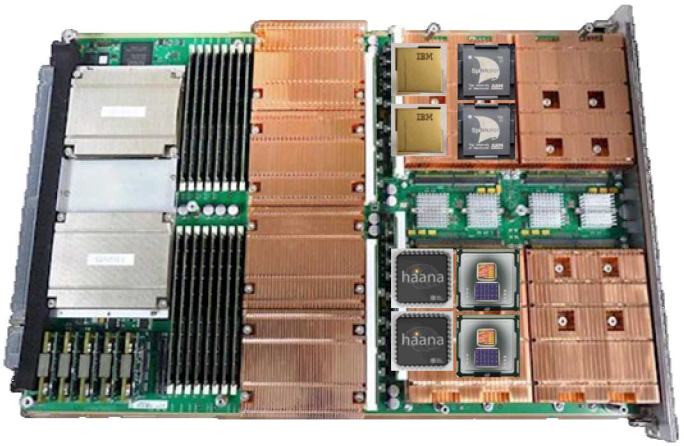
Let's imagine fully integrated neuromorphic onto HPC platforms (sitting next to GPUs and CPUs)

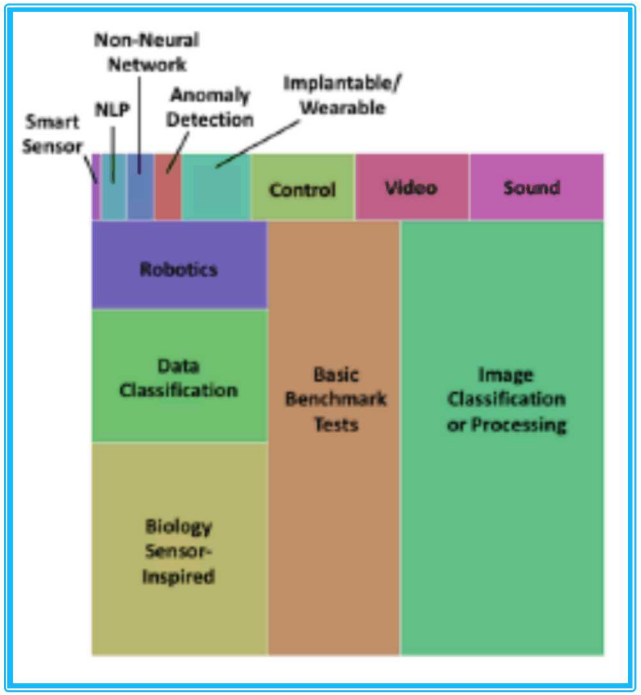


Emerging neuromorphic chips

- Ultra-low power spiking circuits
- Scalable architecture → easy to achieve millions of neurons

Machine Learning is one of many applications





Katie Schuman, ORNL, 2017

Biological-inspired neural algorithms

Machine Learning / Deep Learning

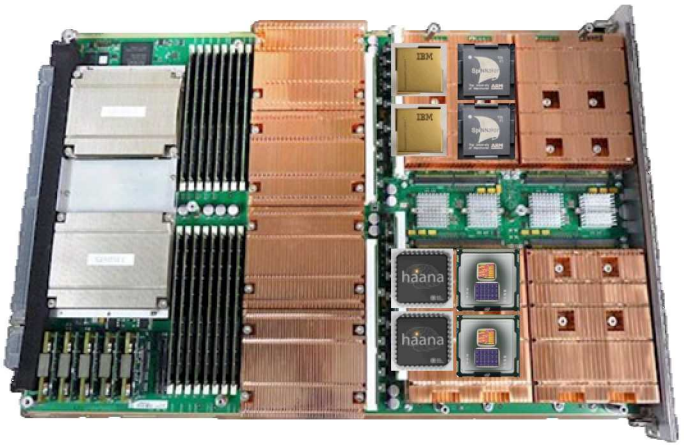
Surrogate Models; Reduced Order Modeling

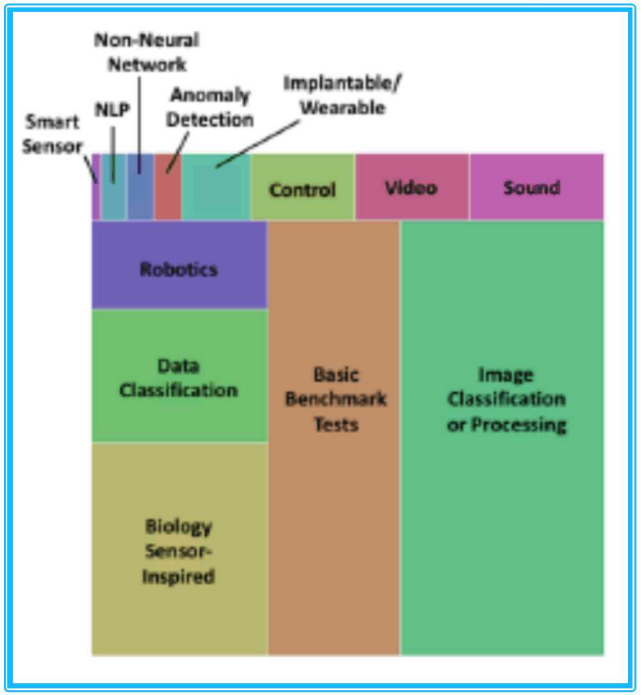
Neural-implemented numerical and scientific computing

Random Walk Methods

Graph Analytics

Machine Learning is one of many applications





Katie Schuman, ORNL, 2017

Biological-inspired neural algorithms

Machine Learning / Deep Learning

Surrogate Models; Reduced Order Modeling

Neural-implemented numerical and scientific computing

Random Walk Methods

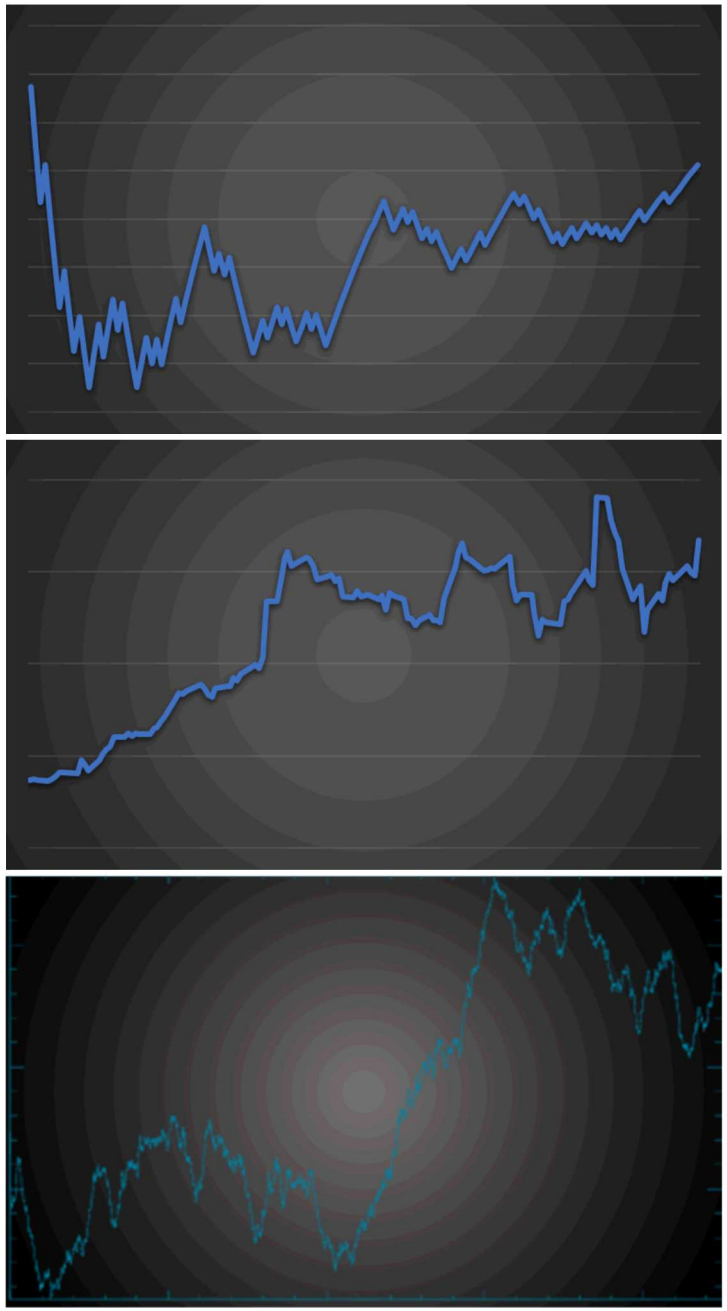
Graph Analytics

Diffusion via Random Walk

- Diffusion can be modeled either as a deterministic PDE or a stochastic process.
  - The diffusion PDE is given by

$$\frac{\partial}{\partial t} f(t,x) = D \frac{\partial^2}{\partial x^2} f(t,x).$$

- To solve via a stochastic process, many random walks are sampled to statistically approximate a solution
  - The mean *density* of walkers approaches the expected mean, equal to the deterministic solution, at a rate of $1/\sqrt{N}$.



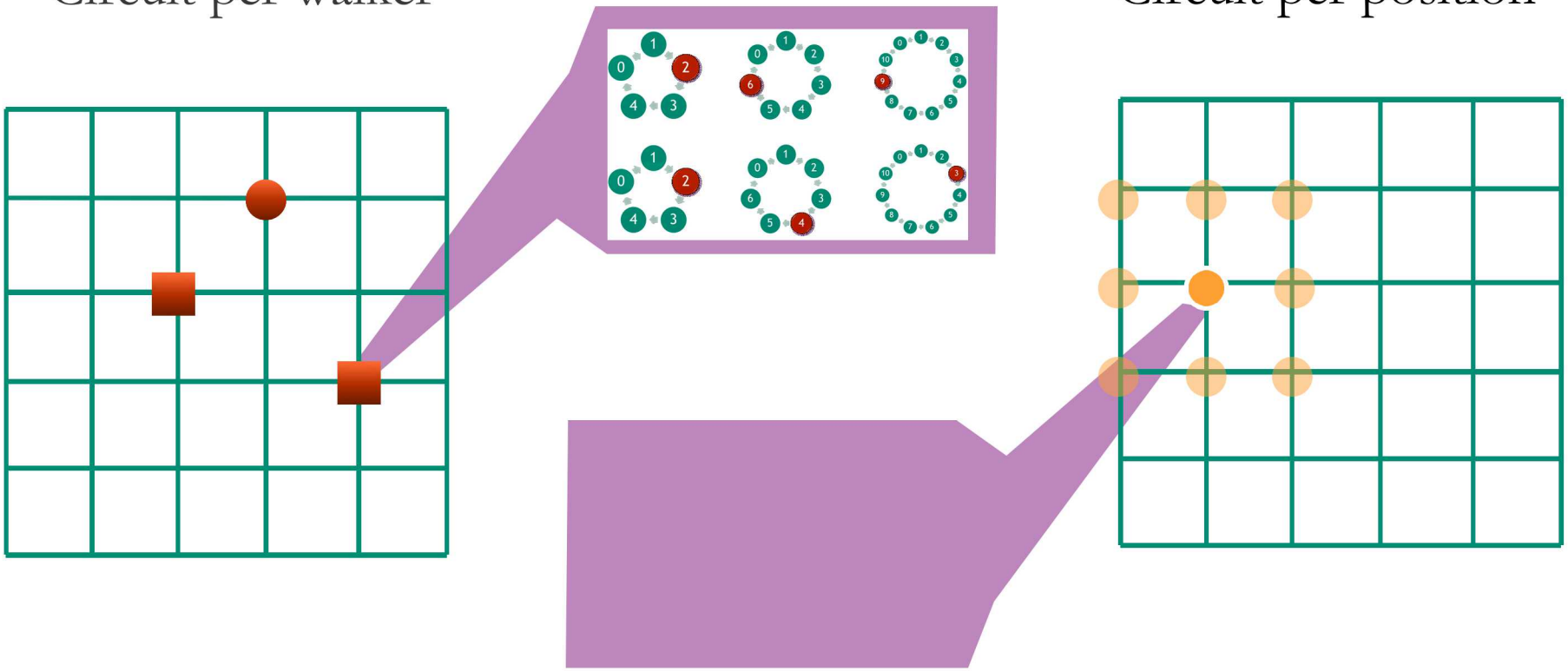
Implementing Random Walks on Spiking Neuromorphic

Particle Method

Circuit per walker

Density Method

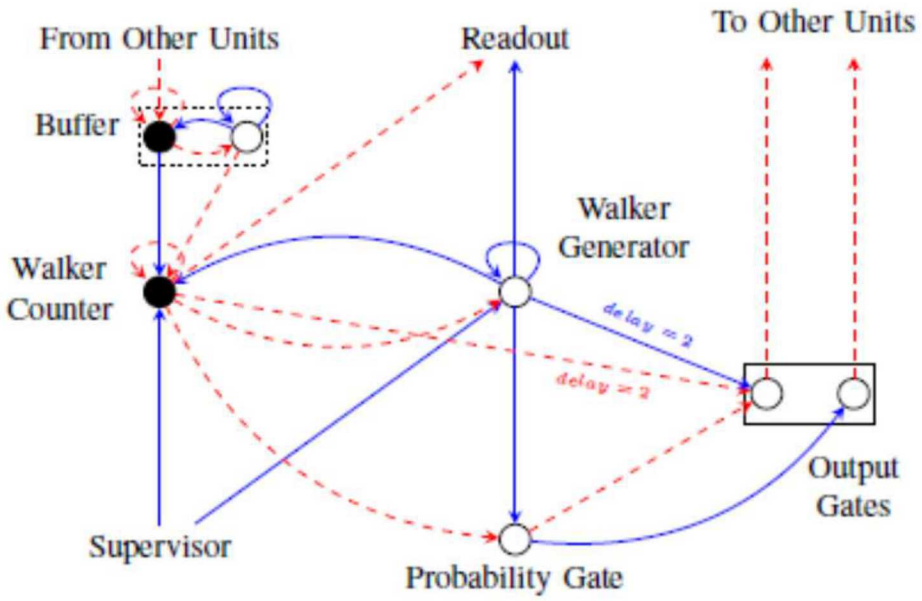
Circuit per position



See Severa, W., et. al, *IJCNN* (2018), IEEE, 1-8.

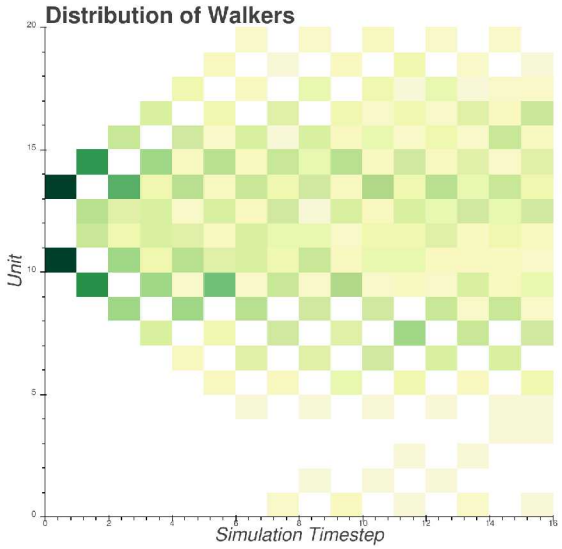
The Density Method

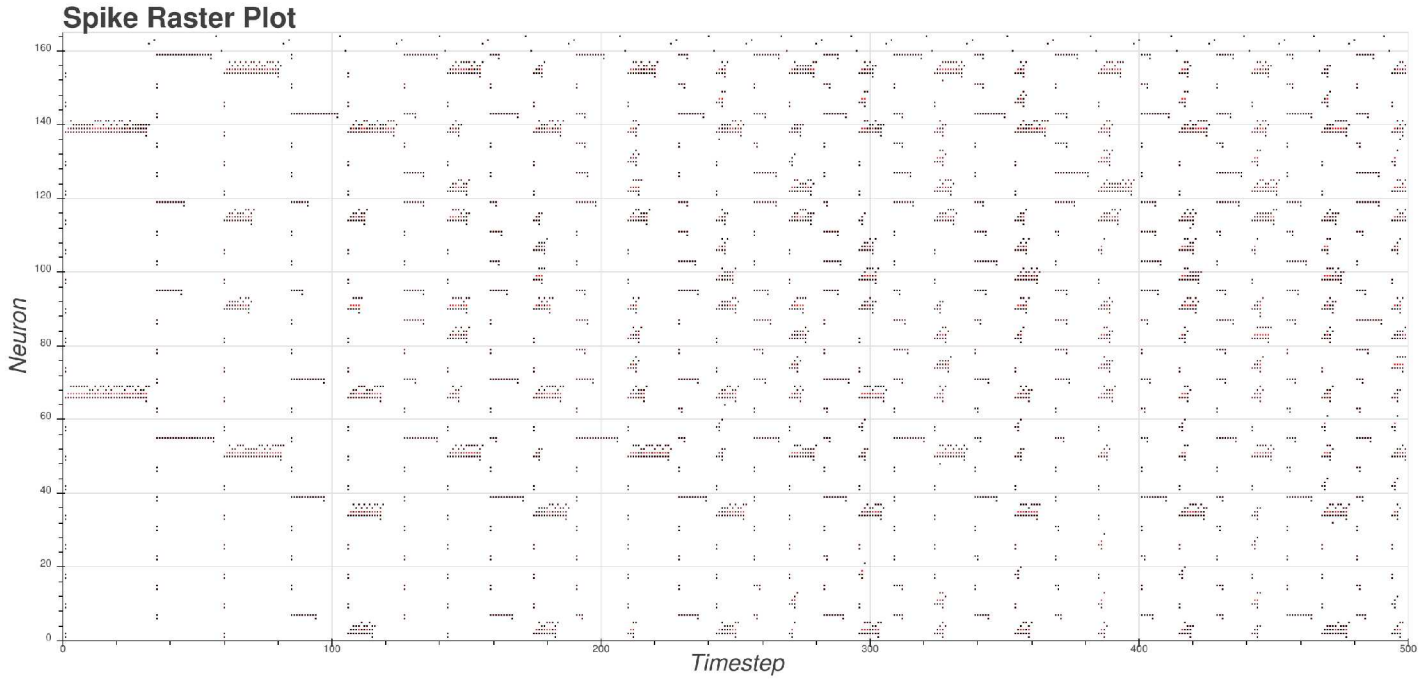
- Each vertex encodes density of particles in the internal potential of certain nodes
- Each time step "hands off" particles to connected vertices according to probabilistic maps



Measure	Cost (for k locations, simulating N walkers; 1-D case)
Walker memory	O(1)
Connection memory	O(k)
Total neurons	O(k)
Time per physical timestep	O(max(ρ_i)), where ρ_i is the density of walkers at each location
Position energy per timestep	O(N)
Update energy per timestep	O(N)

Speed of Simulation Depends on Maximum Density



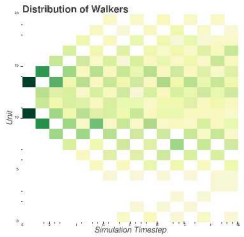


Connecting General IPDEs and Random Walks

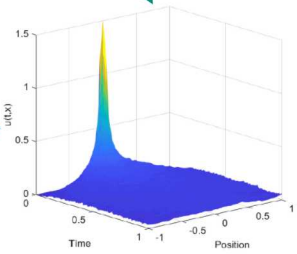
$$u_t = du_{xx}$$

$$u(0,x) = \begin{cases} -150x^2 + \frac{3}{2}, & x \in \left[-\frac{1}{10}, \frac{1}{10}\right] \\ 0, & \text{otherwise} \end{cases}$$

$\mathbb{P}$ Traditional

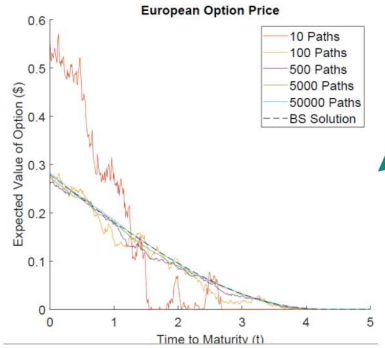


$\mathbb{E}$



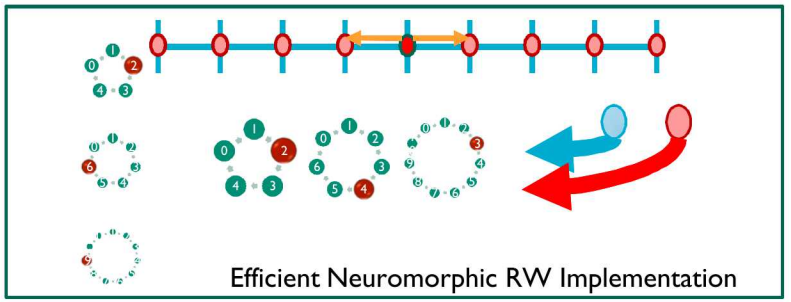
Known

$$-V_t + rV = \frac{1}{2}\sigma^2 x^2 V_{xx} + rxV_x$$



Deduction from underlying stock process

Challenging Applications, Low Power Implementation



+

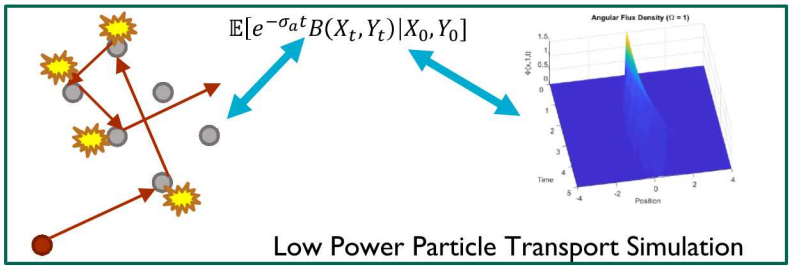
$$\mathrm{d}X_t = \mu(t, X_t)\mathrm{d}t + \sigma(t, X_t)\mathrm{d}W_t$$

$$u_t = f(u_x, u_{xx}, t)$$

$$u(t,x) = \mathbb{E}[g(X_t)|X_0 = x]$$

Rigorous Probabilistic Connections

=



A Class of Integro-PDEs with a Probabilistic Interpretation

The IPDE-IVP

$$u_t(t,\boldsymbol{x}) = \frac{1}{2}\sum_{i,j} a_{i,j}(t,\boldsymbol{x})u_{x_i x_j}(t,\boldsymbol{x}) + \sum_i b_i(t,\boldsymbol{x})u_{x_i}(t,\boldsymbol{x})$$
$$+\lambda(t,\boldsymbol{x})\int \Big(u\big(t,\boldsymbol{x}+h(t,\boldsymbol{x},q)\big) - u(t,\boldsymbol{x})\Big)\phi_Q(q;t,\boldsymbol{x})\mathrm{d}q$$
$$+c(t,\boldsymbol{x})u(t,\boldsymbol{x}) + f(t,\boldsymbol{x})$$
$$u(t,\boldsymbol{x}) = g(\boldsymbol{x})$$

has solution

$$u(t,\boldsymbol{x}) = \mathbb{E}\left[g(\boldsymbol{X}_t)\exp\left(\int_0^t c(s,\boldsymbol{X}_s)ds\right) + \int_0^t f(s,\boldsymbol{X}_s)\exp\left(\int_0^s c(u,\boldsymbol{X}_u)du\right)ds \,\middle|\, \boldsymbol{X}_0 = \boldsymbol{x}\right]$$

where

$$\mathrm{d}\boldsymbol{X}_t = b(t,\boldsymbol{X}_t)\mathrm{d}t + \sigma(t,\boldsymbol{X}_t)\mathrm{d}W_t + h(t,\boldsymbol{X}_t,Q)\mathrm{d}P_{t;,Q,t,\boldsymbol{X}_t}$$

and a, b, c, g, h, and f are all real valued, $\lambda < 0$; further for each t and $\boldsymbol{x}$ that $\phi_Q \geq 0$ and $\int \phi_Q(q)\mathrm{d}q$ so that $P(t;Q,t,\boldsymbol{x})$ is a Poisson process with rate $-\int_0^t \lambda(s,\boldsymbol{x})\mathrm{d}s$. We further require that $a = \sigma\sigma^\top$, b, and h are all defined so that the stochastic process $\boldsymbol{X}_t$ has a unique solution that belongs almost surely to the domain of g.

A Class of Integro-PDEs with a Probabilistic Interpretation

The IPDE-IVP

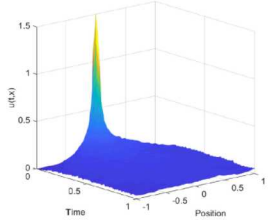
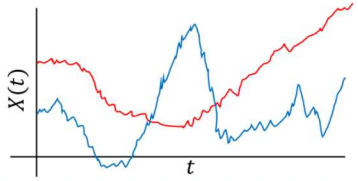
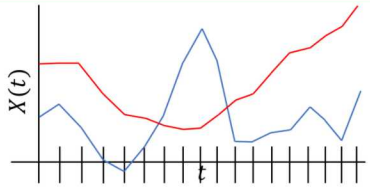
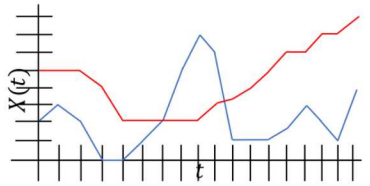
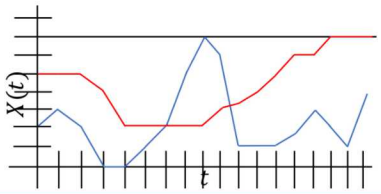
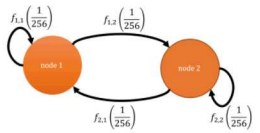
$$u_t(t,\boldsymbol{x}) = \frac{1}{2}\sum_{i,j} a_{i,j}(t,\boldsymbol{x})u_{x_i x_j}(t,\boldsymbol{x}) + \sum_i b_i(t,\boldsymbol{x})u_{x_i}(t,\boldsymbol{x})$$
$$+\lambda(t,\boldsymbol{x})\int \Big(u\big(t,\boldsymbol{x}+h(t,\boldsymbol{x},q)\big) - u(t,\boldsymbol{x})\Big)\phi_Q(q;t,\boldsymbol{x})\mathrm{d}q$$
$$+c(t,\boldsymbol{x})u(t,\boldsymbol{x}) + f(t,\boldsymbol{x})$$
$$u(t,\boldsymbol{x}) = g(\boldsymbol{x})$$

$$u(t,\boldsymbol{x}) = \mathbb{E}\left[g(\boldsymbol{X}_t)\exp\left(\int_0^t c(s,\boldsymbol{X}_s)ds\right) + \int_0^t f(s,\boldsymbol{X}_s)\exp\left(\int_0^s c(u,\boldsymbol{X}_u)du\right)ds \,\middle|\, \boldsymbol{X}_0 = \boldsymbol{x}\right]$$

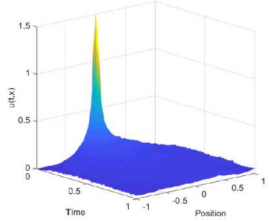
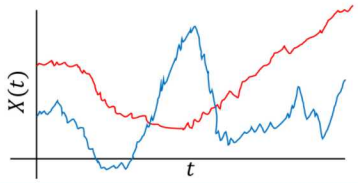
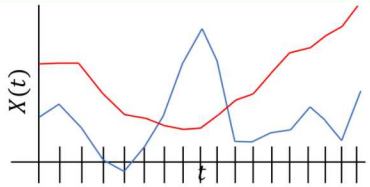
Non-Zero Terms	Application	SDE Example
a	Heat Equation	$\mathrm{d}X_t = \sigma\mathrm{d}W_t$
a, b, f	European Option Pricing	$\mathrm{d}S_t = rS_t\mathrm{d}t + \sigma S_t\mathrm{d}W_t$
b, λ, h, c, f	Particle Transport	$dX_t = -vY_t dt;\ \ dY_t = \omega_{Y_t} dP_{t;Y_t}$
a, f	Electrostatic Scalar Potential*	$\mathrm{d}X_t^{(i)} = \sqrt{\varepsilon}\mathrm{d}W_t$
b, c	Pollutant Source Deterioration	$\mathrm{d}X_t = v\mathrm{d}t^{\wedge}$

*steady state solution, ^actually an ODE

Accuracy Stack for Neuromorphic Implementation

PDE Ground Truth	$u_t = f(t, u, u_x, u_{xx})$ $u(t,x) = \mathbb{E}[g(t, X_t) \mid X_0 = x]$		
Problem	**Approximation**	**Visualization**	**Error/Convergence**
To approximate the expectation, we must sample paths of the stochastic process.	$u(t,x) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(t, X_t^i\right);\ X_0^{(i)} = x$		$\frac{1}{\sqrt{M}}$
Continuous paths cannot be sampled, we must employ a discretization scheme.	$u(j\Delta t, x) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(j\Delta t, X_{j\Delta t}^i\right);\ X_0^{(i)} = x$		$\sqrt{\Delta t}$
Neurons cannot represent a continuum of locations. Hence we must limit the spatial locations of the walk.	$u(j\Delta t, x_k) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(j\Delta t, \hat{X}_{j\Delta t}^i\right);\ \hat{X}_0^{(i)} = x_k$		$\frac{1}{2} j\Delta t \Delta s$
There are a finite number of neurons, so maximum and minimum values for the random walk will exist.	$u(j\Delta t, x_k) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(j\Delta t, \check{X}_{j\Delta t}^i\right);\ \check{X}_0^{(i)} = x_k$		varies
Hardware Specific Issues. TrueNorth having quantized probability, for example.	$\mathbb{P} \propto \frac{1}{256}$		varies

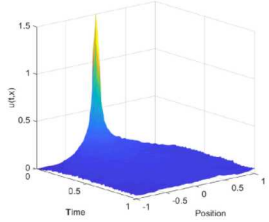
Accuracy Stack for Neuromorphic Implementation

PDE Ground Truth	$u_t = f(t, u, u_x, u_{xx})$ $u(t,x) = \mathbb{E}[g(t, X_t) \mid X_0 = x]$		
Problem	**Approximation**	**Visualization**	**Error/Convergence**
To approximate the expectation, we must sample paths of the stochastic process.	$u(t,x) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(t, X_t^i\right);\ X_0^{(i)} = x$		$\frac{1}{\sqrt{M}}$
Continuous paths cannot be sampled, we must employ a discretization scheme.	$u(j\Delta t, x) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(j\Delta t, X_{j\Delta t}^i\right);\ X_0^{(i)} = x$		$\sqrt{\Delta t}$

⇧ ⇧ ⇧ ⇧ ⇧ ⇧

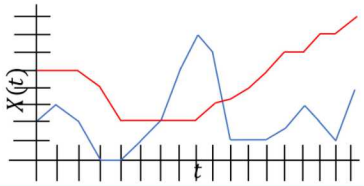
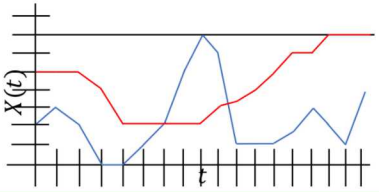
Present for any implementation

Accuracy Stack for Neuromorphic Implementation

Problem	Approximation	Visualization	Error/Convergence
PDE Ground Truth	$u_t = f(t, u, u_x, u_{xx})$ $u(t,x) = \mathbb{E}[g(t, X_t) \mid X_0 = x]$		

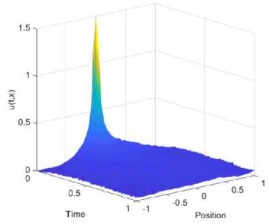
Neuromorphic Specific

⇩ ⇩ ⇩ ⇩ ⇩ ⇩

Neurons cannot represent a continuum of locations. Hence we must limit the spatial locations of the walk.	$u(j\Delta t, x_k) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(j\Delta t, \hat{X}^i_{j\Delta t}\right);\ \ \hat{X}^{(i)}_0 = x_k$		$\frac{1}{2} j\Delta t \Delta s$
There are a finite number of neurons, so maximum and minimum values for the random walk will exist.	$u(j\Delta t, x_k) \approx \frac{1}{M}\sum_{i=1}^{M} g\left(j\Delta t, \check{X}^i_{j\Delta t}\right);\ \ \check{X}^{(i)}_0 = x_k$		varies

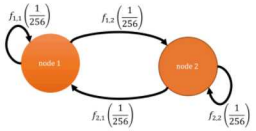
⇧ ⇧ ⇧ ⇧ ⇧ ⇧

Accuracy Stack for Neuromorphic Implementation

Problem	Approximation	Visualization	Error/Convergence
PDE Ground Truth	$u_t = f(t, u, u_x, u_{xx})$ $u(t,x) = \mathbb{E}[g(t, X_t) \mid X_0 = x]$		

Hardware Specific

⇩ ⇩ ⇩ ⇩ ⇩ ⇩

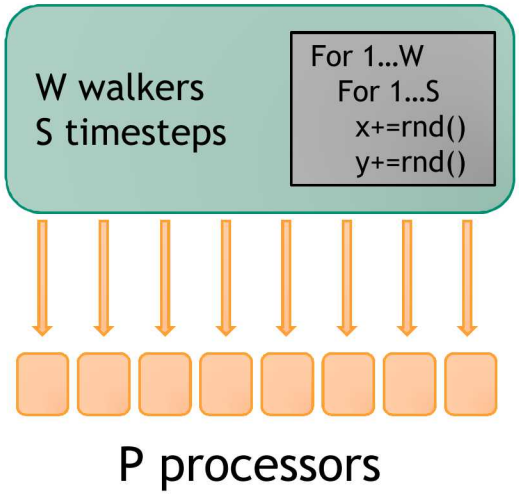
Hardware Specific Issues. TrueNorth having quantized probability, for example.	$\mathbb{P} \propto \frac{1}{256}$		varies

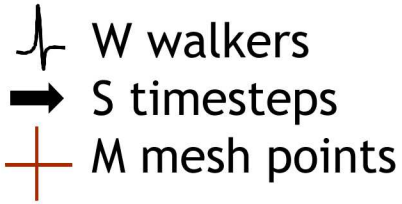
Analysis of Random Walk Algorithm

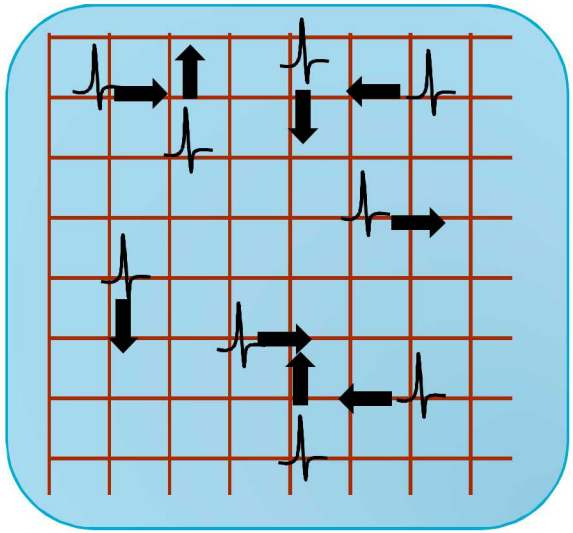
Goal: Can we get an "apples to apples" complexity analysis of two algorithm versions (normal vs density)?

Approach: Consider time / power costs of simulating a number of walkers, *W*, over a number of time steps, *S*. Conventional CPUs can use however many processors are on the chip (*P*), and neuromorphic chips can use however many mesh points (*M*) that can fit on the chip.

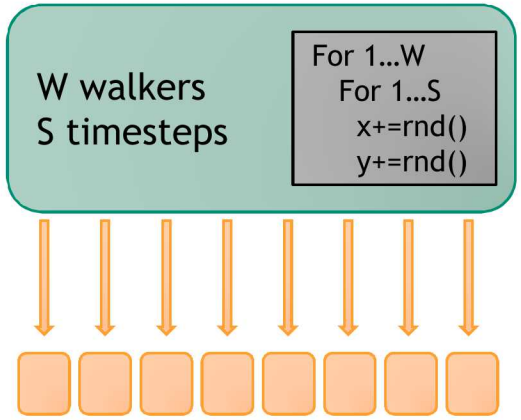
Problem: Simple 2D local diffusion







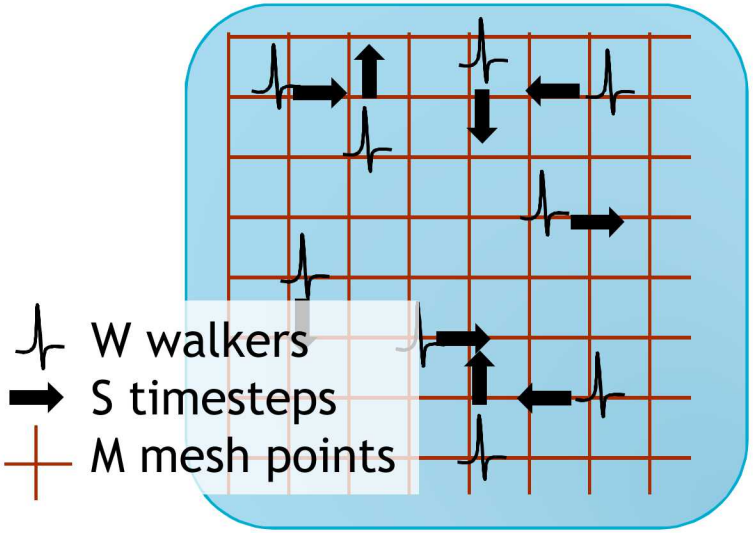
Analysis of Random Walk Algorithm



Conventional Analysis

$$T = c_{CPU,time}\frac{W * S}{P}$$

$$P_{ower} = c_{CPU,power} W * S$$



Neural Analysis

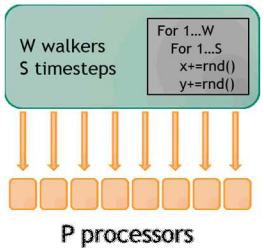
$$T = c_{Neural,time}\frac{W * S}{kM}$$

k is a function of the highest average walker density mesh point relative to average density

$$P_{ower} = c_{Neural,power} W * S$$

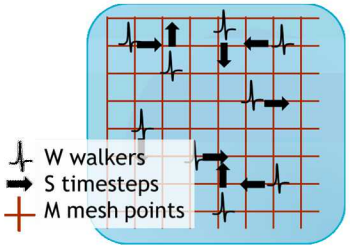
Analysis of Random Walk Algorithm

Conventional Analysis



$$T = c_{CPU,time}\frac{W * S}{P}$$

$$Power = c_{CPU,power} W * S$$



Neural Analysis

$$T = c_{Neural,time}\frac{W * S}{kM}$$

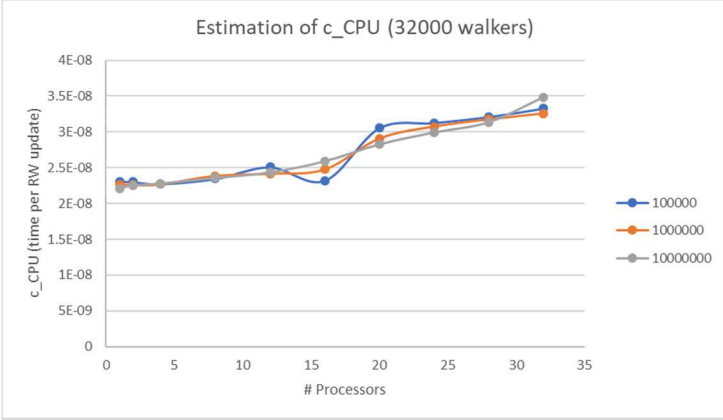
$$Power = c_{Neural,power} W * S$$

Time assessment:
Generally, $k*M >> P$. So if $c_{Neural,\ time} \leq c_{CPU,\ time}$, then the neural chip will be faster.

For applications where k<<1 (average walker distribution is highly skewed, e.g.) the *time* benefit of neural computing will decrease.

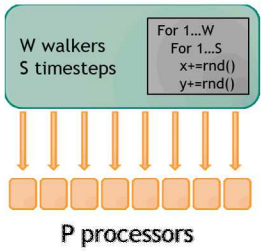
Empirical question:
Assess $c_{Neural,\ time}$ and $c_{CPU,\ time}$ for a modern CPU and a neuromorphic chip from above equations.

- Preliminary estimates performed for CPU using C++/OpenMP.
- **Need estimates for TrueNorth or Loihi (ideally both).**



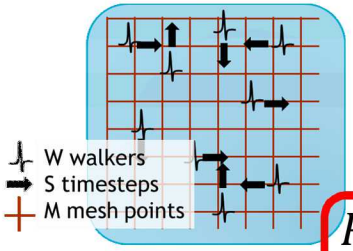
Analysis of Random Walk Algorithm

Conventional Analysis



$$T = c_{CPU,time}\frac{W * S}{P}$$

$$Power = c_{CPU,power} W * S$$



Neural Analysis

$$T = c_{Neural,time}\frac{W * S}{kM}$$

$$Power = c_{Neural,power} W * S$$

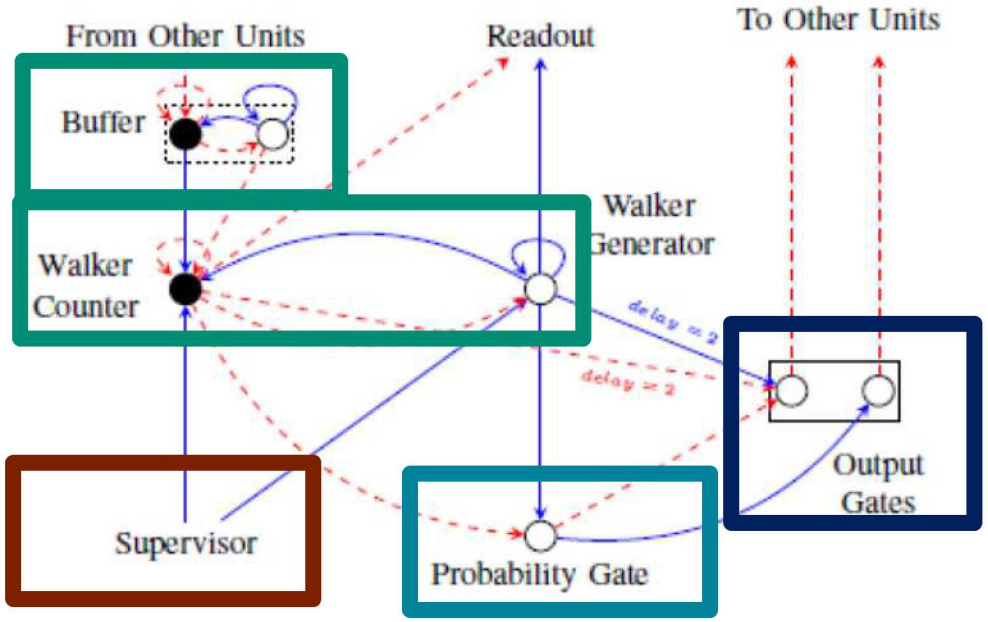
Power assessment:
For CPUs, there is likely no inherent advantage of parallelization for power.
Efficiency of walker distribution (k) is not likely to affect power (in event-driven hardware).
However, there is a strong likelihood that $c_{Neural,\ power} << c_{CPU,\ power}$

Empirical question:
Can we assess $c_{Neural,\ power}$ and $c_{CPU,\ power}$ for a modern CPU and a neuromorphic chip from above equations?

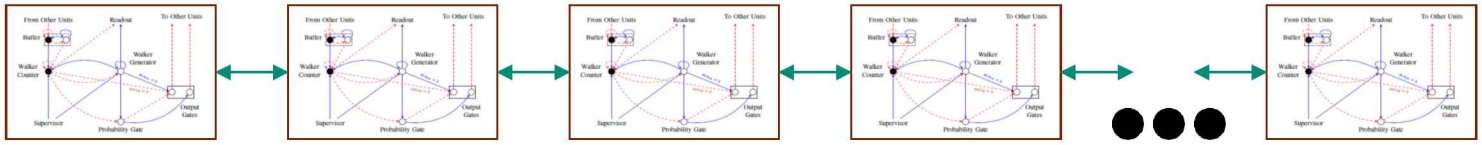
- A coarse assessment can likely be made from power specifications of chips for 100% operation over idle.
- Can we build off of preliminary estimates performed for CPU using C++/OpenMP?
- **Need estimates for TrueNorth or Loihi (ideally both).**

Monte Carlo on Loihi

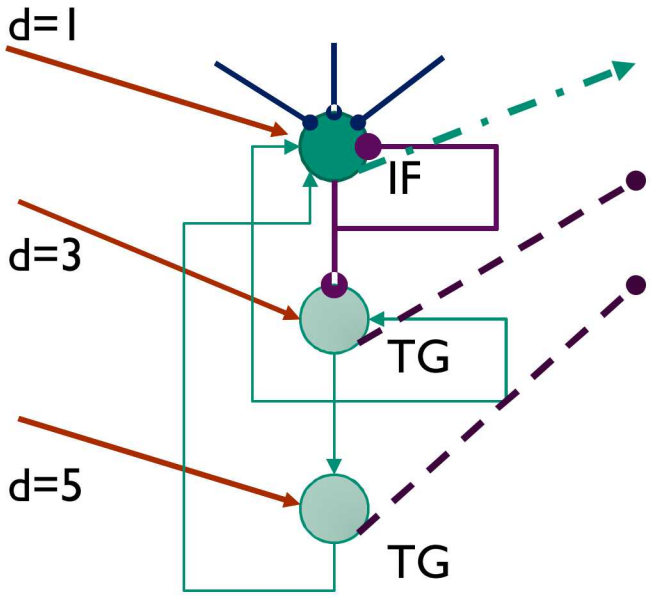
Loihi-specific circuit



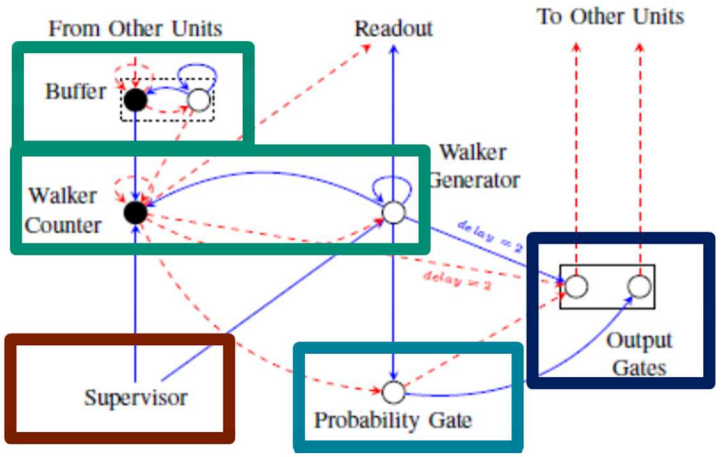
1. Supervisor circuit
   1. Start buffer
   2. Start counter
2. Counter circuit
   1. Buffer neurons
   2. Counter neurons
3. Probabilistic neurons
4. Output neurons

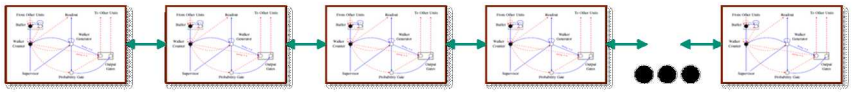


Loihi-specific circuit – node buffer and node counter



To next counter circuit or output nodes



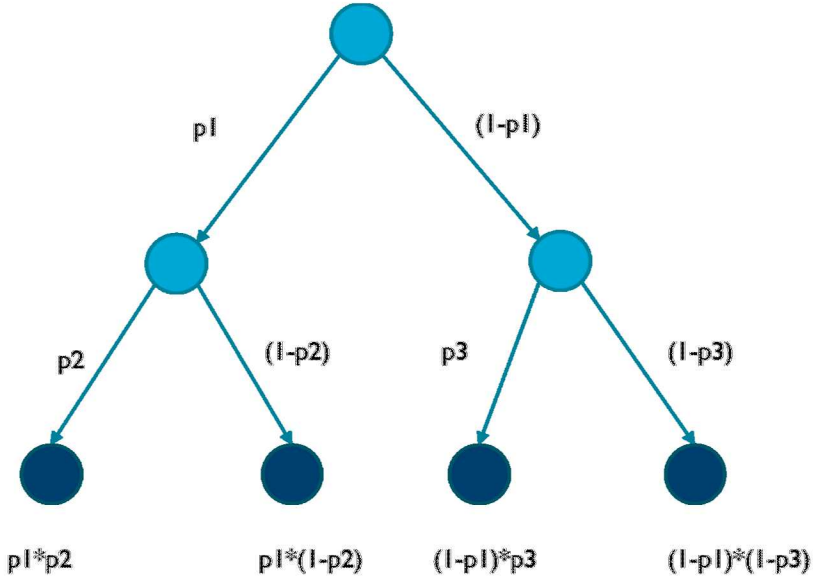


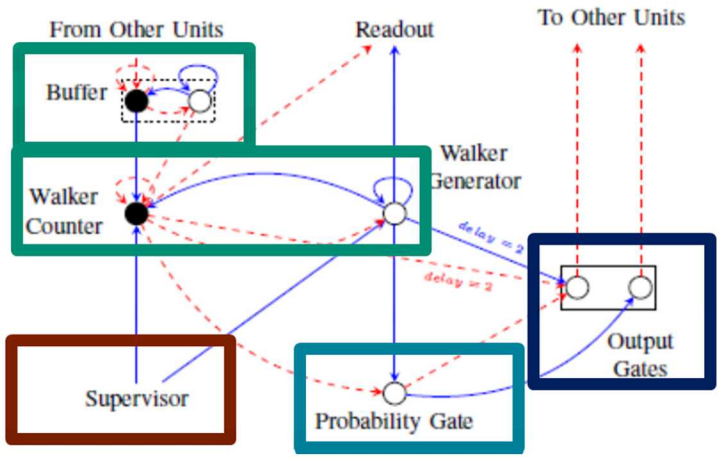
1- COUNTER neuron stores number of walkers as *negative voltage*

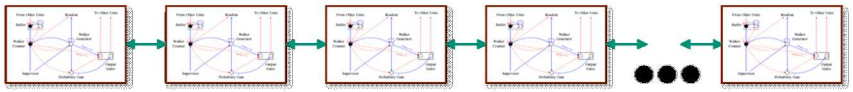
2- Supervisor input causes GENERATOR to fire as long as $V_{COUNTER}$ is negative

3- RELAY neuron ensures that $V_{COUNTER}$ is appropriately reset if it pre-emptively shuts off

Loihi-specific circuit – random probabilities and outputs



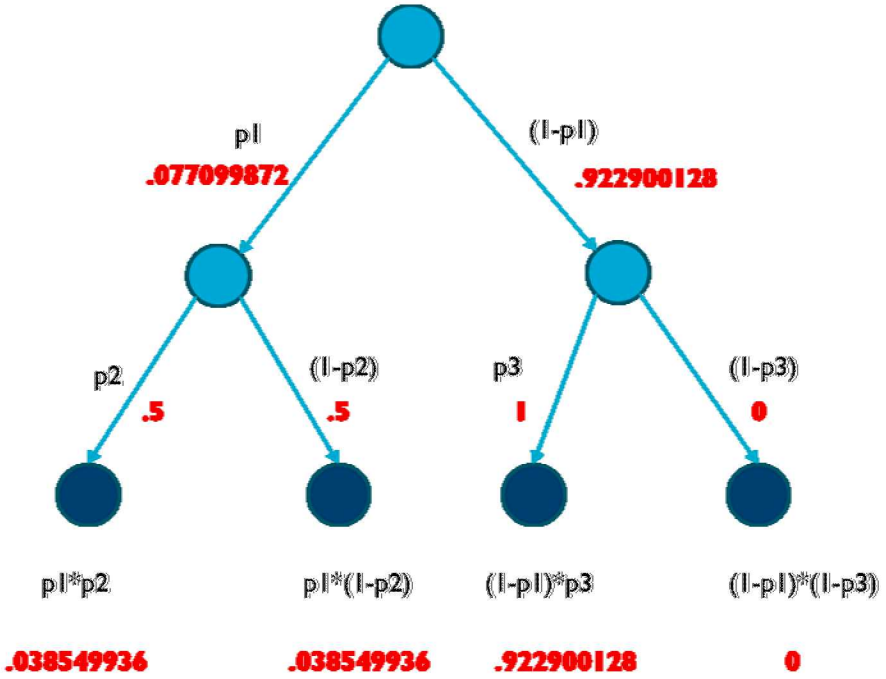


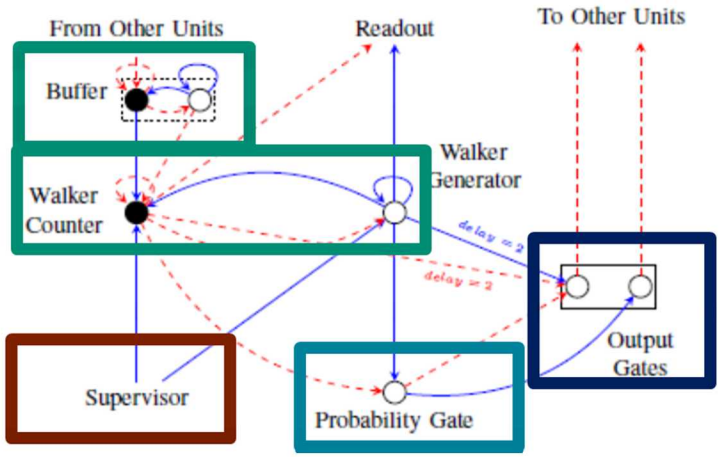


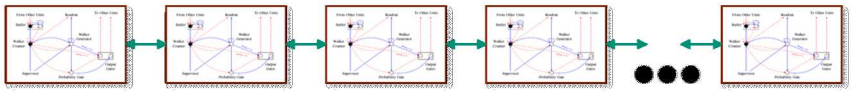
1- Goal: we need to get pre-defined probabilities that the walker gets directed to along the kth output direction

2 – In principle, a tree structure would work, but it does not scale well

Loihi-specific circuit – random probabilities and outputs





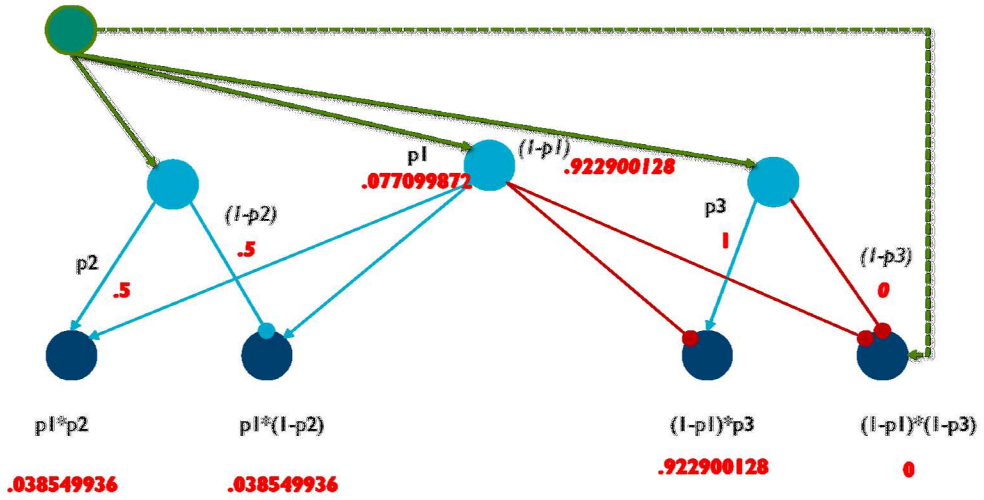


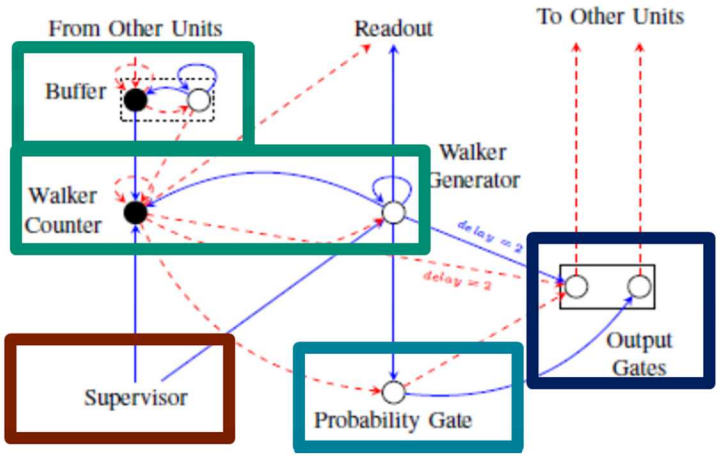
1- Goal: we need to get pre-defined probabilities that the walker gets directed to along the kth output direction

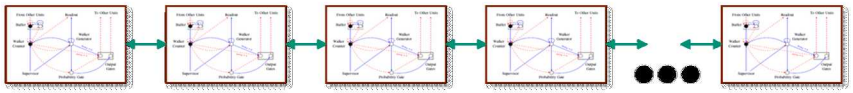
2 – In principle, a tree structure would work, but it does not scale well

- Can use tree to compute probabilities

Loihi-specific circuit – random probabilities and outputs







1- Goal: we need to get pre-defined probabilities that the walker gets directed to along the kth output direction

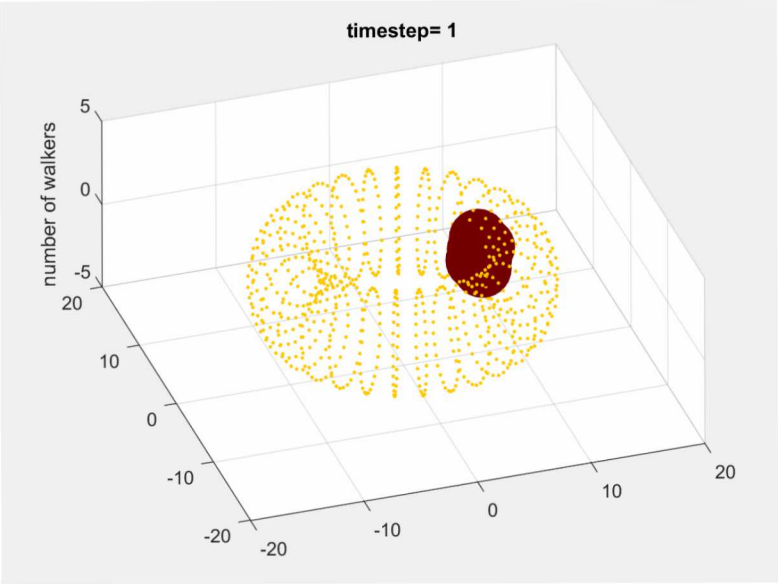
2 – In principle, a tree structure would work, but it does not scale well

- Can use tree to compute probabilities

3 – Can collapse tree into single layer

- Single layer tree is both faster and has predictable delays…

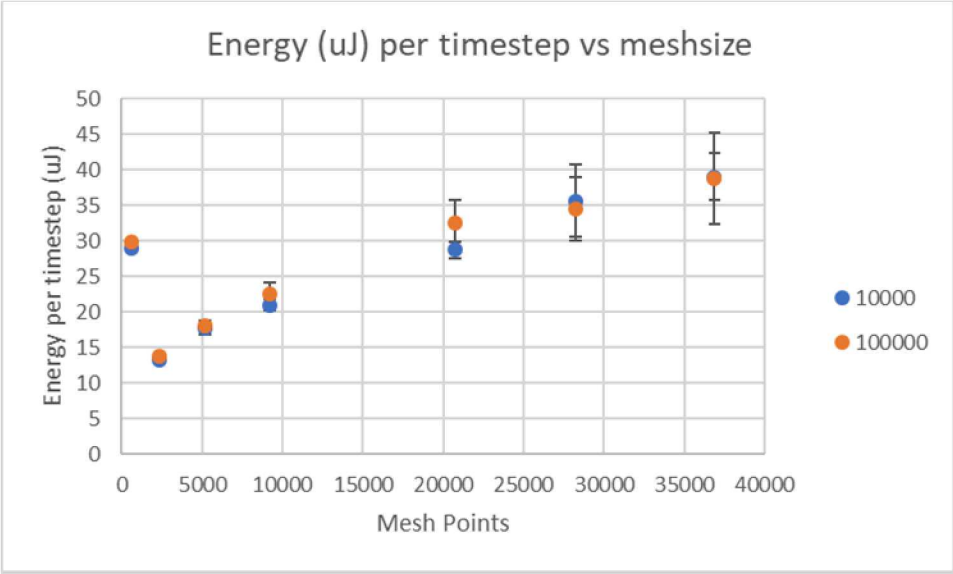
Task 1: Simple diffusion

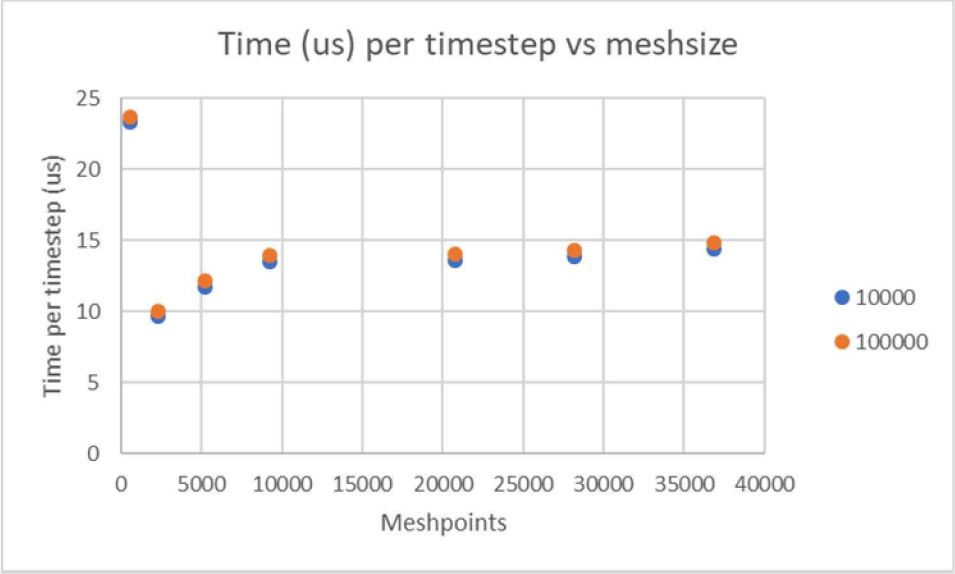


- 30 x 30 torus mesh
- 40 Loihi cores
- 600 random walkers
- 10,000 network timesteps (152 model timesteps

- This is really as big as we can go due to cap on # of probes

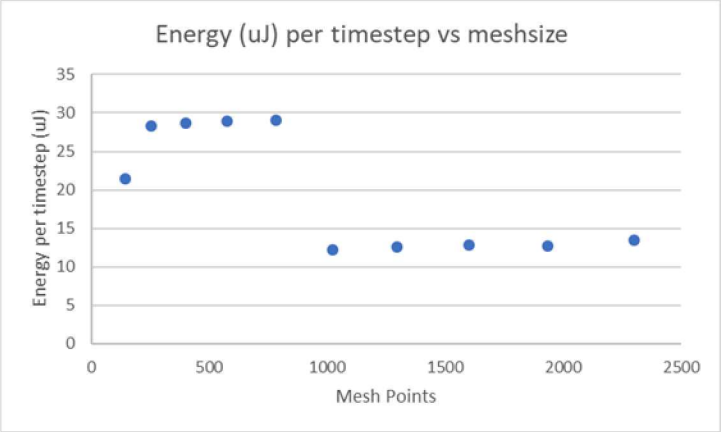
We can't see results, but we can characterize them…

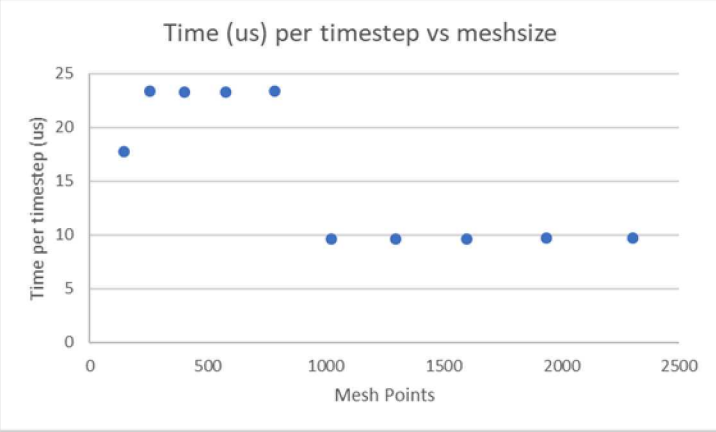




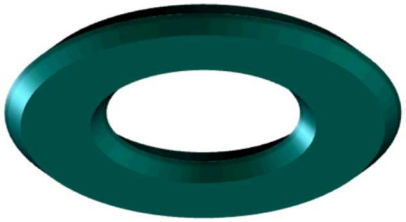
Some uncertainty around Loihi

- Small models seem to have added expense / overhead
- Certain mesh-sizes don't run / compile (probably my fault)
- 3 chips fails. Other chip counts are okay
- Haven't figured out how to track all mesh points (yet)

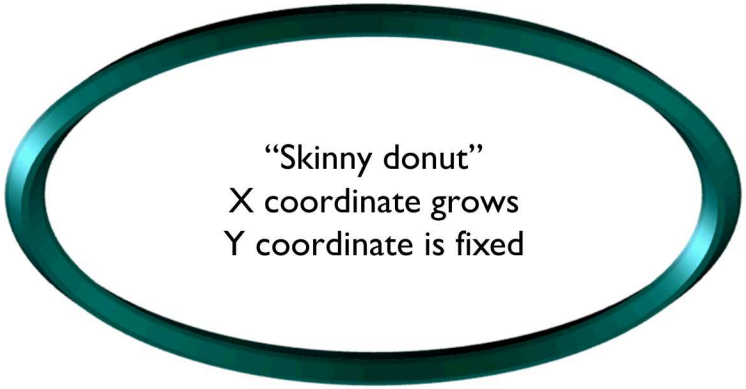




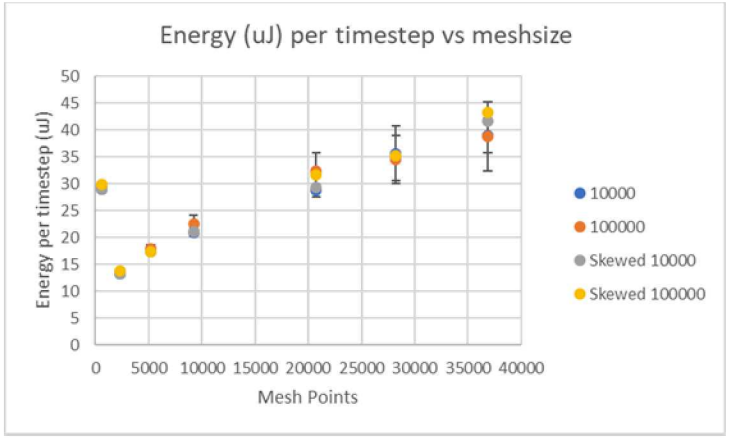
Loihi does run similarly with different shape simulations

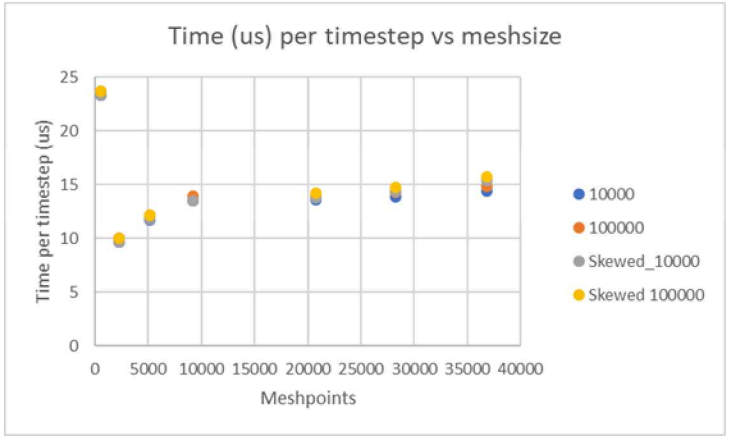


"Fat donut"
X,Y coordinates the same size

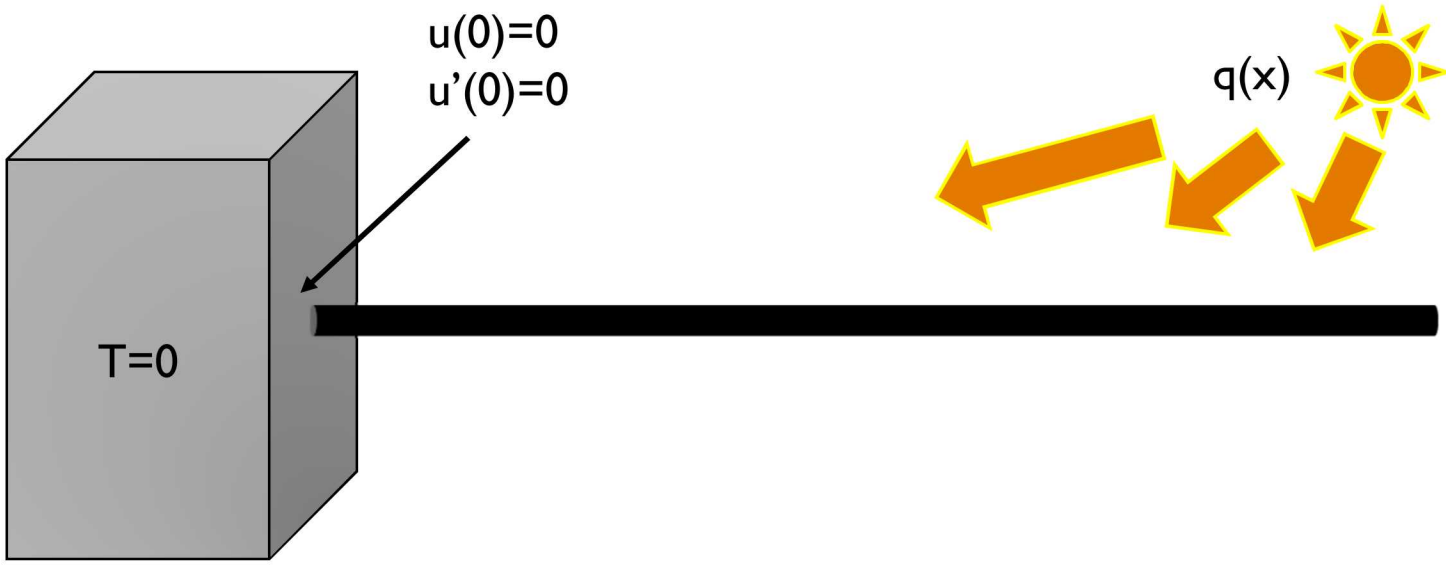


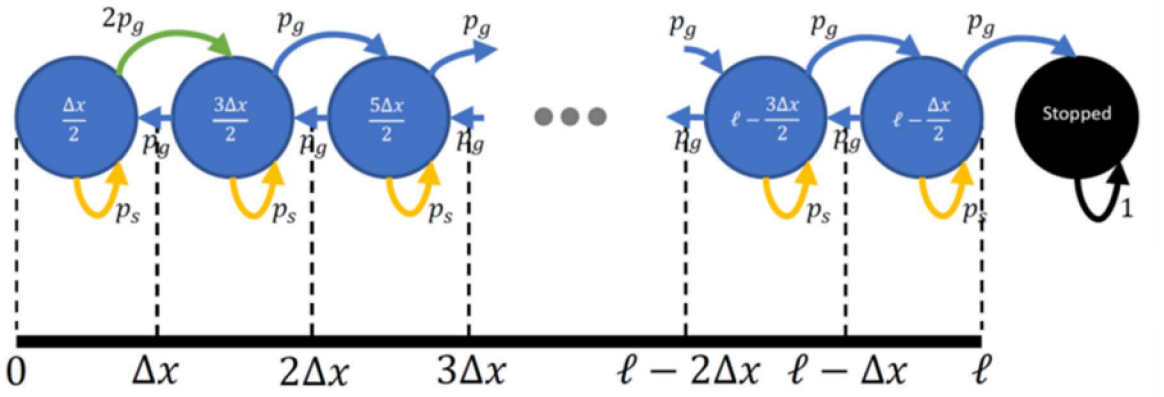
"Skinny donut"
X coordinate grows
Y coordinate is fixed





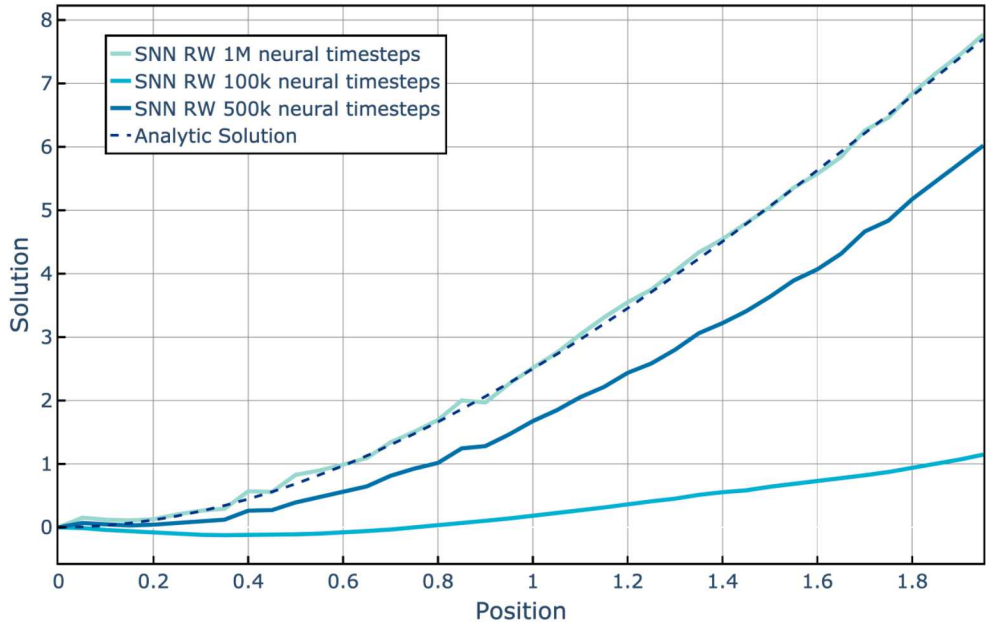
Task 2: A real problem: Steady-state PDE solution of heat





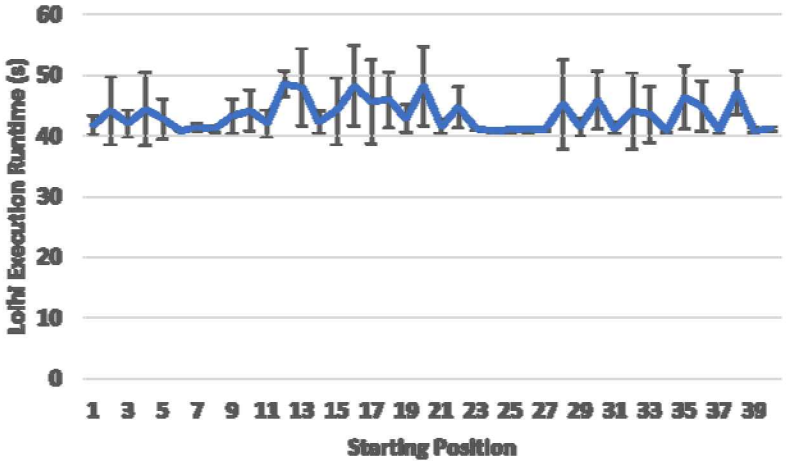
Well-suited for neuromorphic implementation

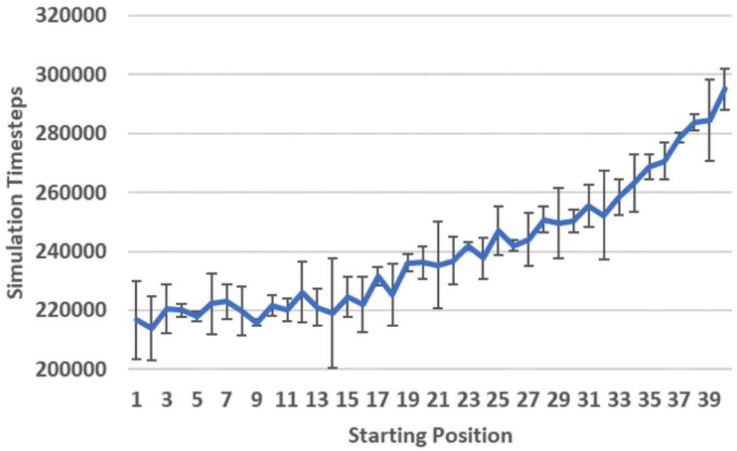
- Long time scale - walkers run until they are absorbed by end-node
  - Can be a very long time to completion
  - For 40 mesh points, walkers require ~250,000 simulation timesteps on average to fall off
  - Steady-state approximation hurt by cutting off early
  - Simulation timesteps are not directly related to neural timesteps
- Requires a lot of walkers
  - Typical for Monte Carlo approaches



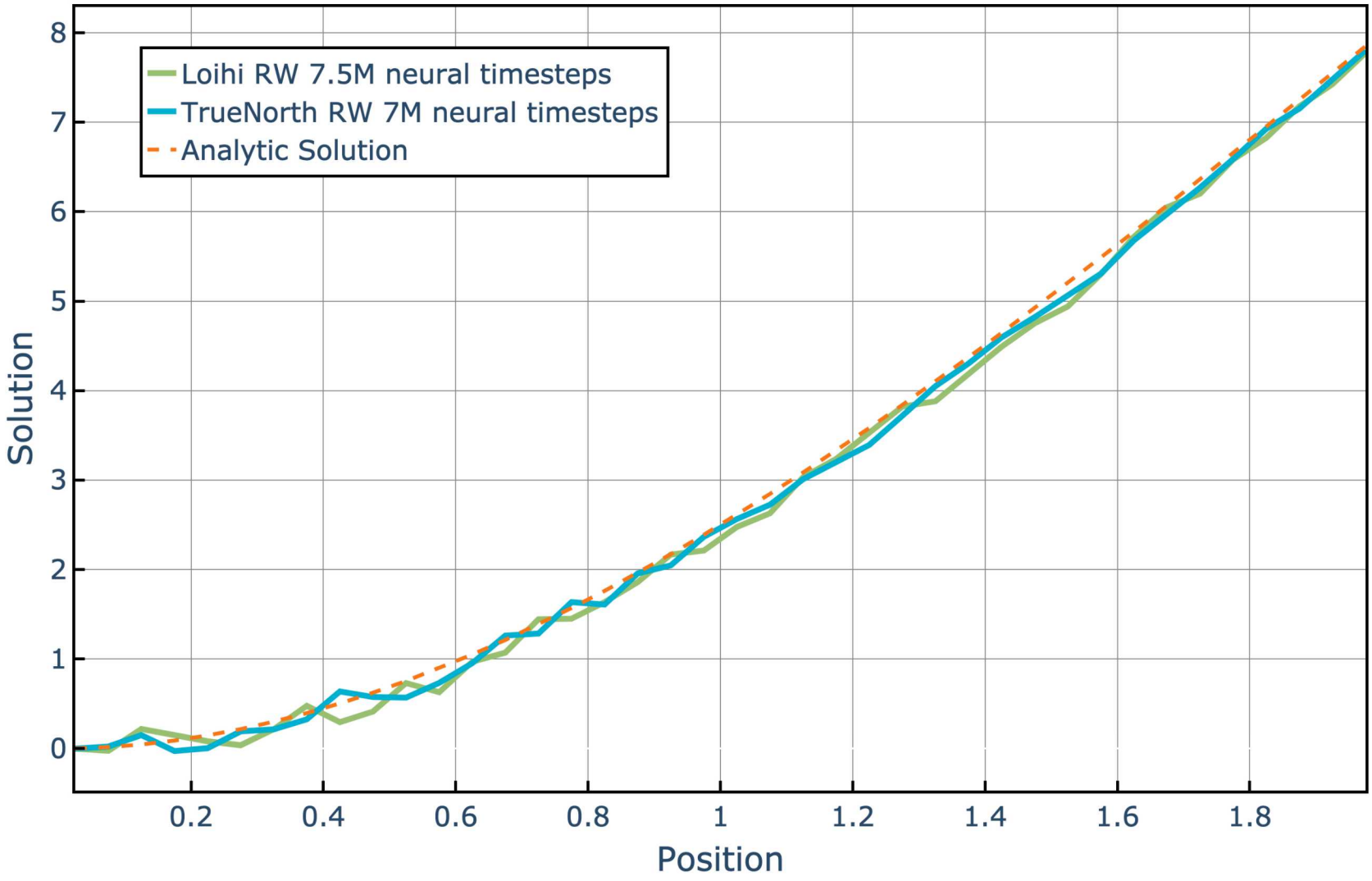
Highlights of Loihi Results

- 10,000 walkers requires 40 runs of 250 walkers each, for each of the 40 wire locations
- Simulations run for 7,500,000 timesteps. On average between 200,000 and 300,000 simulation timesteps
- Runs very quickly: ~40 seconds for 7.5 M timesteps
- Only ~13 neurons per mesh point; in principle could put many copies on Nahuku board
  - Limitation becomes readout probes





Loihi results are close to perfect

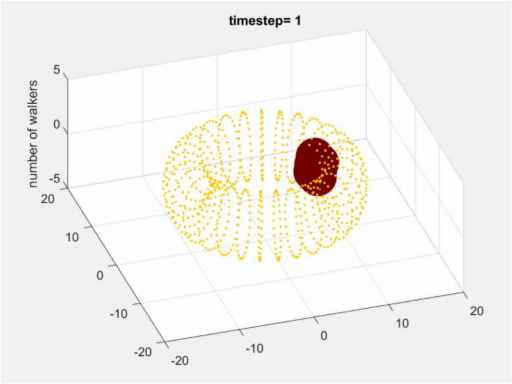


How far can this go?

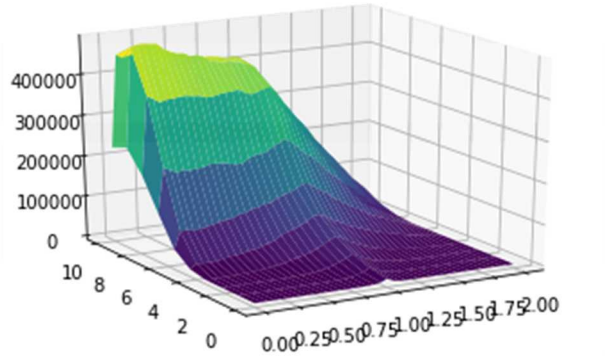
$$\frac{\partial}{\partial t} f(t,x) = D \frac{\partial^2}{\partial x^2} f(t,x).$$

$$u_t(t,x) = \frac{1}{2}\sum_{i,j} a_{i,j}(t,x) u_{x_i x_j}(t,x) + \sum_i b_i(t,x) u_{x_i}(t,x)$$
$$+\lambda(t,x) \int \Big(u\big(t, x + h(t,x,q)\big) - u(t,x)\Big) \phi_Q(q;t,x)\mathrm{d}q$$
$$+c(t,x)u(t,x) + f(t,x)$$
$$u(t,x) = g(x)$$

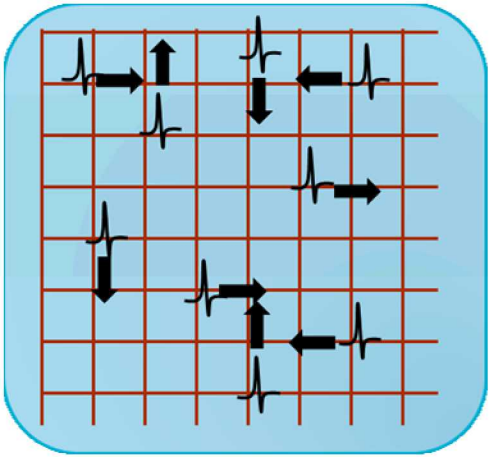
???



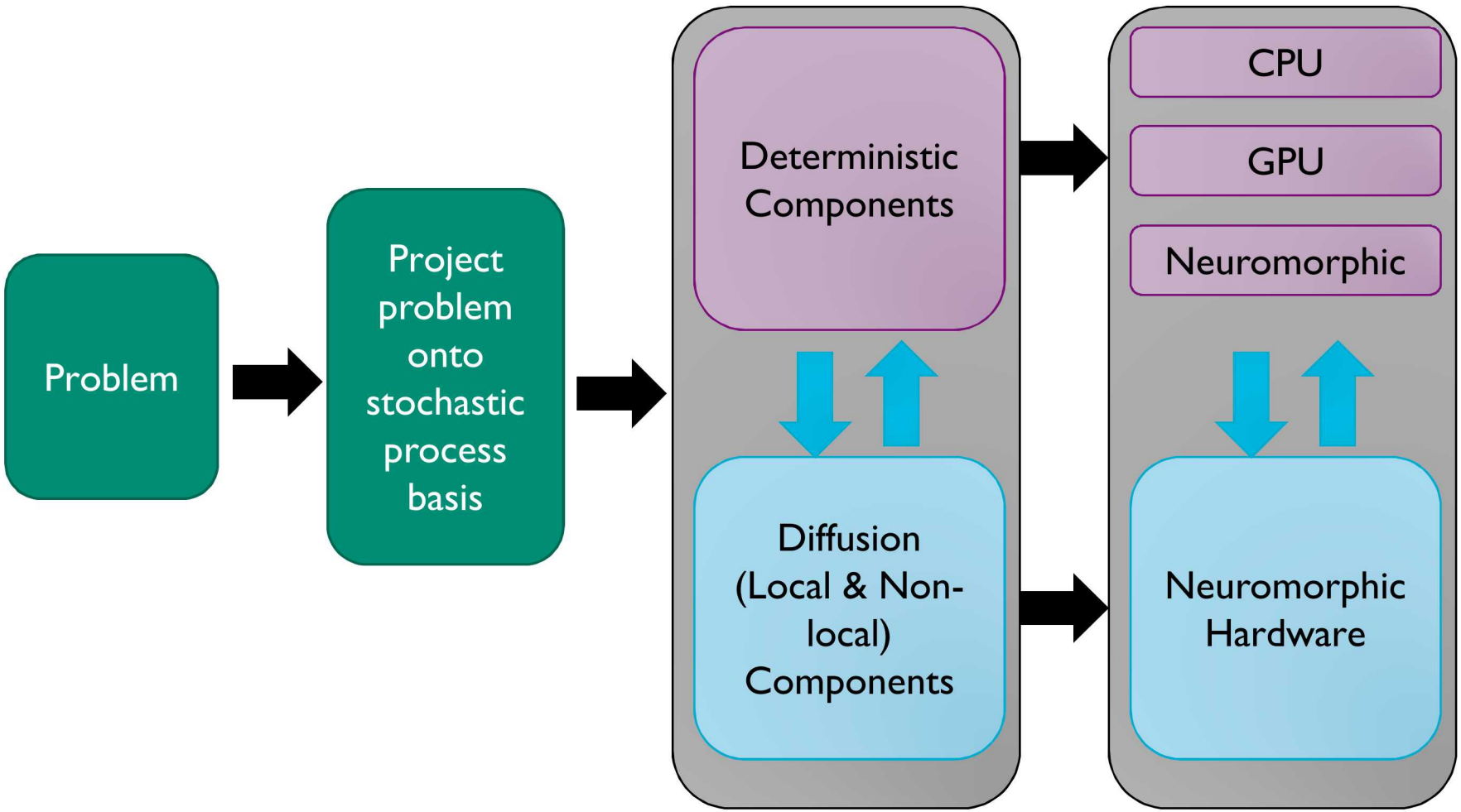
Directly model diffusion



Diffusion process used as component to estimate solution to more complex PDEs

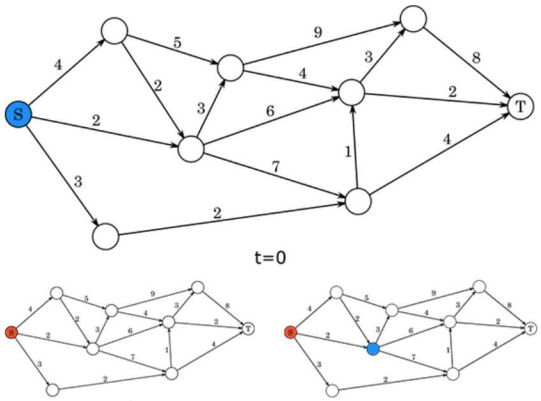


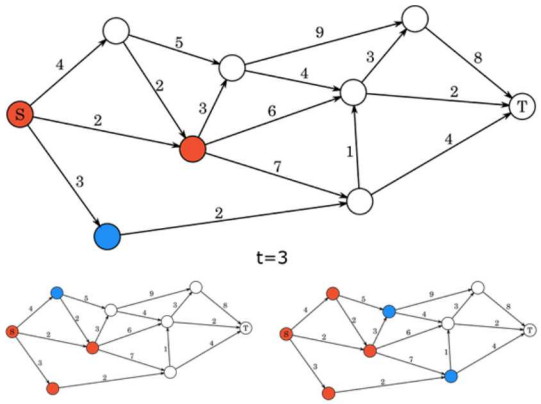
What else can we compute on graphs in parallel?

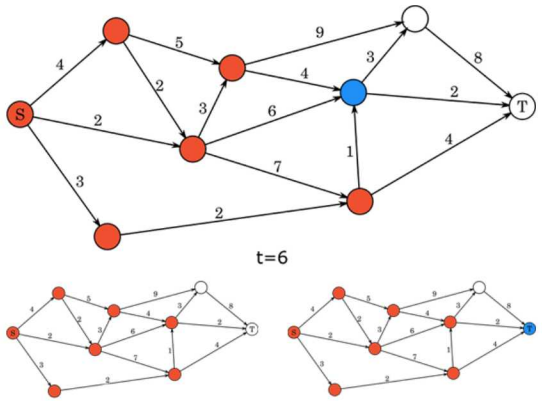

Problem
Project problem onto stochastic process basis
Deterministic Components
Diffusion (Local & Non-local) Components
CPU
GPU
Neuromorphic
Neuromorphic Hardware

Graph Analytics in Spiking Networks

- Growing area of interest
  - Neural circuits are fundamentally graphs…
  - Hamilton et al., 2018; Parekh et al., 2019
- Simple shortest path search is an obvious illustration



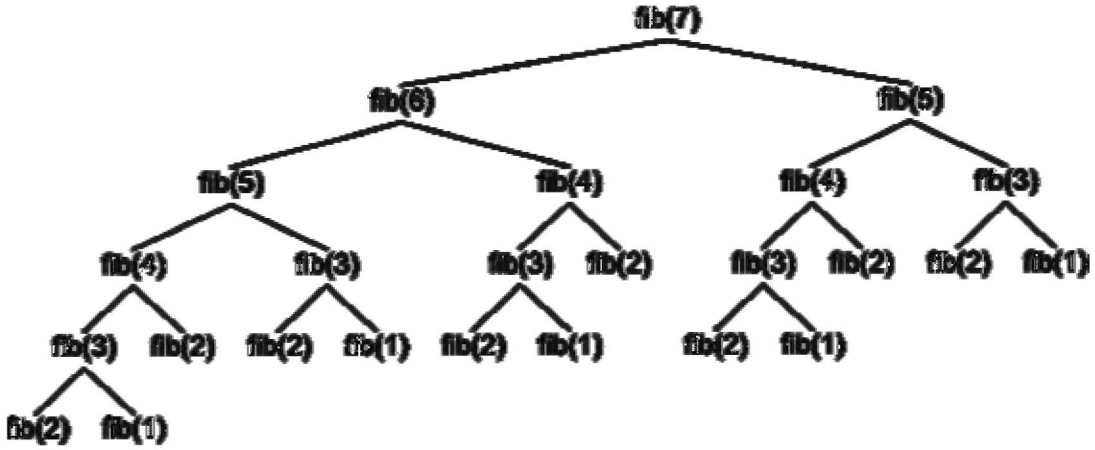


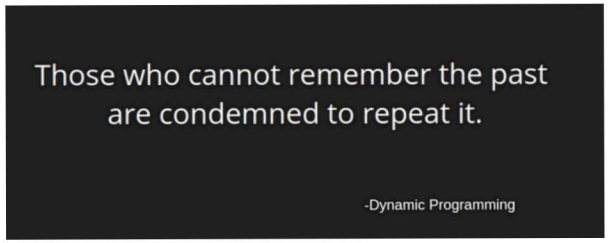


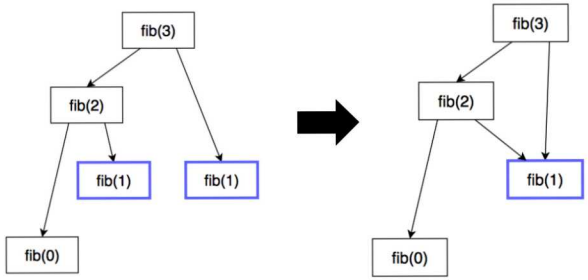
Dynamic Programming

Dynamic programming is a *general technique* for solving certain kinds of discrete optimization problems
Dynamic programming consolidates redundant computation

$$fib(n) = fib(n-1) + fib(n-2); fib(1) = 1, fib(2) = 1$$







[https://blog.usejournal.com/top-50-dynamic-programming-practice-problems-4208fed71aa3]
[https://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html]
[https://medium.com/@shmuel.lotman/the-2-00-am-javascript-blog-about-memoization-41347e8fa603]

Broad Applications of Dynamic Programming

Dynamic programming is a *general technique* for solving certain kinds of discrete optimization problems

- Recurrent solutions to lattice models for protein-DNA binding
- Backward induction as a solution method for finite-horizon discrete-time dynamic optimization problems
- Method of undetermined coefficients can be used to solve the Bellman equation in infinite-horizon, discrete-time, discounted, time-invariant dynamic optimization problems
- Many string algorithms including longest common subsequence, longest increasing subsequence, longest common substring, Levenshtein distance (edit distance)
- Many algorithmic problems on graphs can be solved efficiently for graphs of bounded treewidth or bounded clique-width by using dynamic programming on a tree decomposition of the graph.
- The Cocke–Younger–Kasami (CYK) algorithm which determines whether and how a given string can be generated by a given context-free grammar
- Knuth's word wrapping algorithm that minimizes raggedness when word wrapping text
- The use of transposition tables and refutation tables in computer chess
- The Viterbi algorithm (used for hidden Markov models, and particularly in part of speech tagging)
- The Earley algorithm (a type of chart parser)
- The Needleman–Wunsch algorithm and other algorithms used in bioinformatics, including sequence alignment, structural alignment, RNA structure prediction
- Floyd's all-pairs shortest path algorithm
- Optimizing the order for chain matrix multiplication
- Pseudo-polynomial time algorithms for the subset sum, knapsack and partition problems
- The dynamic time warping algorithm for computing the global distance between two time series
- The Selinger (a.k.a. System R) algorithm for relational database query optimization
- De Boor algorithm for evaluating B-spline curves
- Duckworth–Lewis method for resolving the problem when games of cricket are interrupted
- The value iteration method for solving Markov decision processes
- Some graphic image edge following selection methods such as the "magnet" selection tool in Photoshop
- Some methods for solving interval scheduling problems
- Some methods for solving the travelling salesman problem, either exactly (in exponential time) or approximately (e.g. via the bitonic tour)
- Recursive least squares method
- Beat tracking in music information retrieval
- Adaptive-critic training strategy for artificial neural networks
- Stereo algorithms for solving the correspondence problem used in stereo vision
- Seam carving (content-aware image resizing)
- The Bellman–Ford algorithm for finding the shortest distance in a graph
- Some approximate solution methods for the linear search problem
- Kadane's algorithm for the maximum subarray problem
- Optimization of electric generation expansion plans in the Wein Automatic System Planning (WASP) package

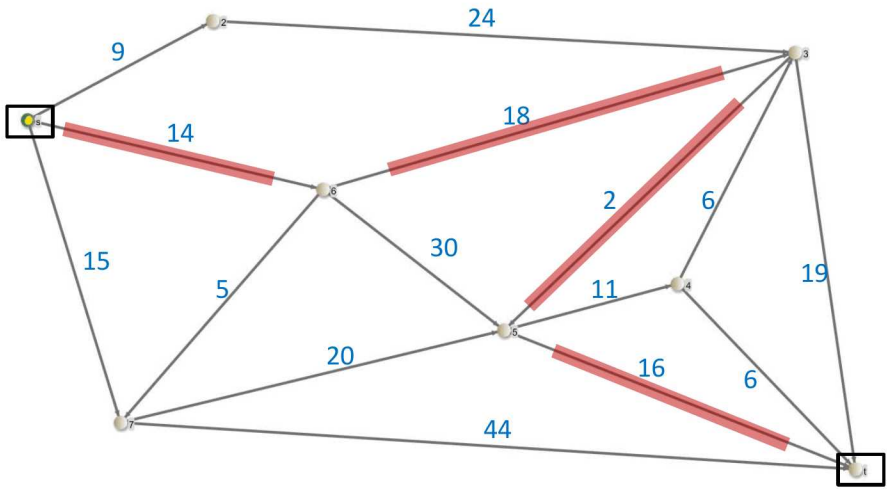
Wikipedia: 30 applications across diverse domains
[https://en.wikipedia.org/wiki/Dynamic_programming]

Another list with 50 applications
[https://blog.usejournal.com/top-50-dynamic-programming-practice-problems-4208fed71aa3]

Spiking Dynamic Programming Approach

New neuromorphic algorithms for dynamic programming
Generically solves a broad class of dynamic programs

Spiking shortest paths algorithm
[Aibara et al., IEEE Int. Symp. on Circuits and Systems, 1991]

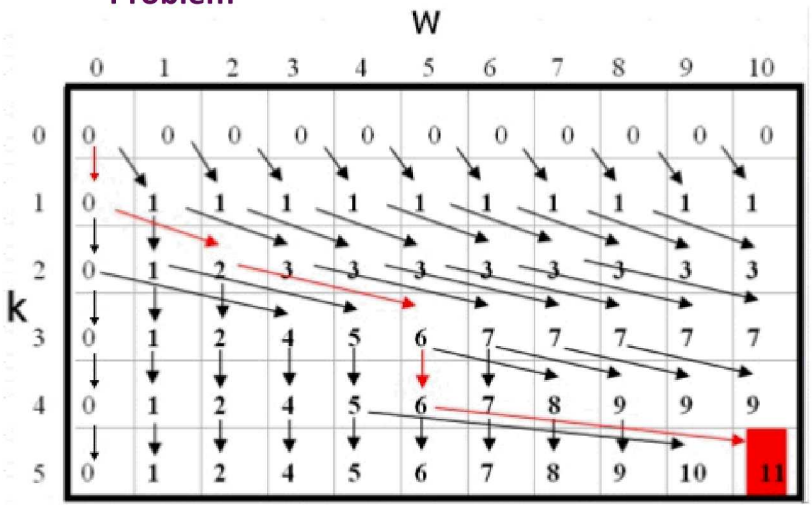


- Our dynamic programming algorithm leverages shortest path NGA
- Single neuron per dynamic program table entry
- Employs delays on links (simulable using recurrent neurons)
- **Novel temporal encoding:** time when neuron first fires represents value of dynamic program table entry

Spiking Dynamic Programming Example

New neuromorphic algorithms for dynamic programming
Spike times encode dynamic programming table values

Dynamic Program for Knapsack Problem



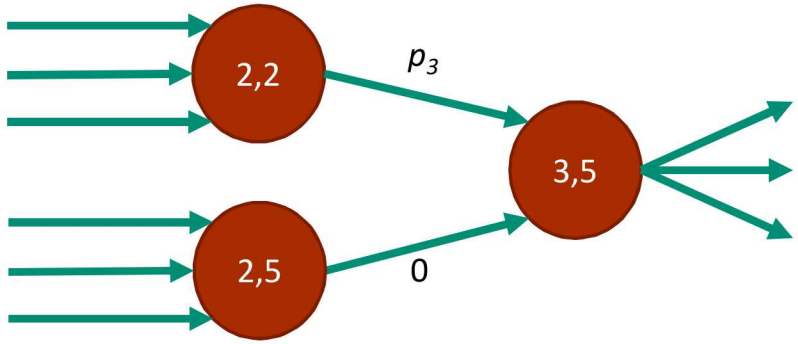
Each table entry is value of best knapsack solution of weight at most W using items {1,…,k}

Knapsack Problem:
N items, each with weight w_i and value v_i

Goal: pick subset of items of weight at most W, maximizing total value.

$$T[3,5] = max\{T[2,5-w_3]+p_3, T[2,5]\}$$

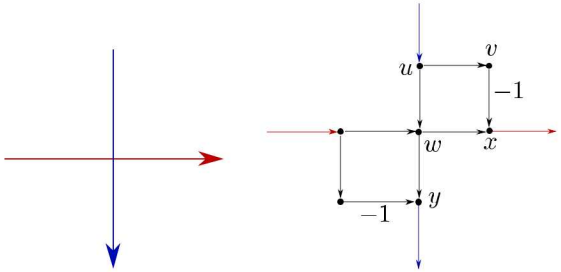
= 6 = 3

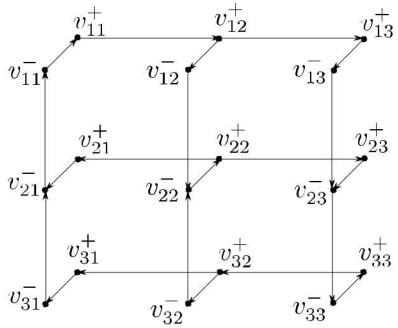


Spiking approach: T[i,j] encoded as time neuron (i,j) receives incoming spike on last of its incoming links

Practical Considerations and Extensions

- Dynamic program graph must be simulated on neuromorphic hardware graph
  New graph embedding problems and techniques
- Neuromorphic hardware has a fixed minimum delay
  Problem-specified delays must be scaled, introducing multiplicative factor to running time
- Dynamic programming graph loading and readout (I/O) costs may present bottlenecks
  Optimized problem-specific algorithms possible (we do so for longest increasing subsequence)
- Spiking approach as presented only gives *value* solution
  Can use O(log n) extra neurons per graph node as memory to store solution
  Novel Hebbian learning approach on edges also works!



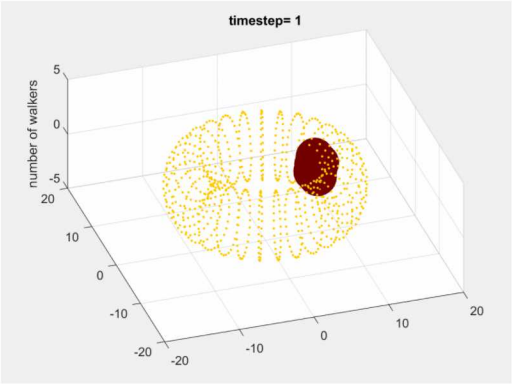


How far can this go?

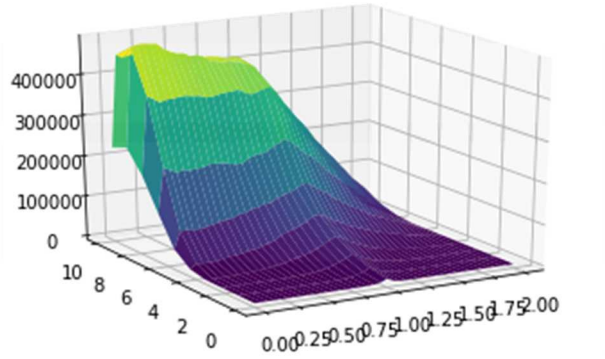
$$\frac{\partial}{\partial t} f(t,x) = D \frac{\partial^2}{\partial x^2} f(t,x).$$

$$u_t(t,x) = \frac{1}{2}\sum_{i,j} a_{i,j}(t,x) u_{x_i x_j}(t,x) + \sum_i b_i(t,x) u_{x_i}(t,x)$$
$$+\lambda(t,x) \int \Big(u\big(t, x + h(t,x,q)\big) - u(t,x)\Big) \phi_Q(q;t,x)\mathrm{d}q$$
$$+c(t,x)u(t,x) + f(t,x)$$
$$u(t,x) = g(x)$$

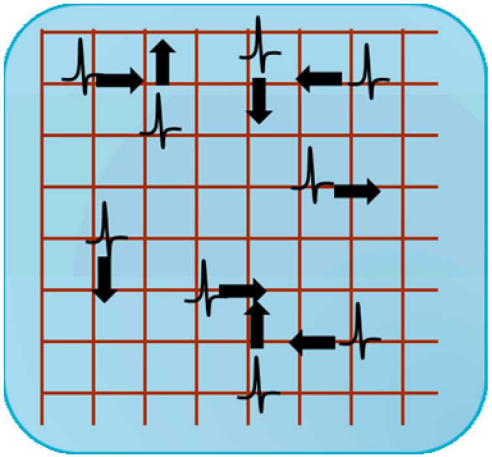
Graph analytics, optimization, …



Directly model diffusion



Diffusion process used as component to estimate solution to more complex PDEs



Many things whose computation can be parallelized onto a graph

Acknowledgments

Neural PDE team:

Darby Smith, William Severa, Aaron Hill, Ojas Parekh, Leah Reeder, Rich Lehoucq, Brian Franke

Neural Graph team:

Ojas Parekh, Cindy Phillips, Ali Pinar, Yipu Wang, Yang Ho, William Severa

Selected References

- Random Walks with Spiking Neuromorphic Hardware
  - 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.
  - Smith et al., ICONS 2020 *submitted*
- Neural Graph Analytics
  - Aimone, J. B., Parekh, O., Phillips, C. A., Pinar, A., Severa, W., & Xu, H. (2019, July). Dynamic Programming with Spiking Neural Computing. In *Proceedings of the International Conference on Neuromorphic Systems* (pp. 1-9).
  - Hamilton, K. E., Mintz, T. M., & Schuman, C. D. (2019). Spike-based primitives for graph algorithms. *arXiv preprint arXiv:1903.10574*.
- Non-AI Applications of Spiking Neuromorphic Hardware
  - 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.
  - 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 Symposium on Parallelism in Algorithms and Architectures,* (2018), ACM, 67-76.
  - 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 Memrstive Neuromorphic Implementation. in *Proeedings of the 7th Annual Neuro-inspired Computation Elements Workshop,* (2019), ACM, 3.
- Generalized Feynman-Kac
  - Grigoriu, M. (2013), *Stochastic Calculus: Applications in Science and Engineering*, Springer Science & Business Media.
- Boltzmann Transport Equation and Neumann Expansion
  - Dupree, S. & Fraley, S. (2002), *A Monte Carlo Primer: A Practical Approach to Radiation Transport*, number v. 1, Springer US.