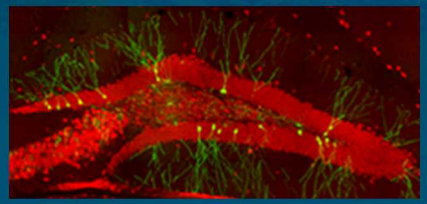
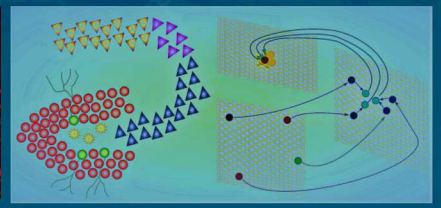
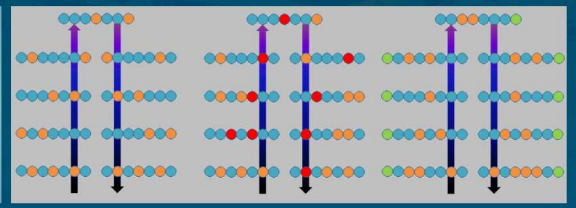
SAND2020-0612C

Neural-Inspired Computation for Efficient Scientific Computing via Random Walks and Surrogate Models







PRESENTED BY

William Severa

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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.

Scientific advancement is requiring ever-increasing compute power

DOE Exascale Computing Project identifies the following challenge areas:

National Security Needs

Advances in Material Science

New Energy Solutions

Advances in Healthcare

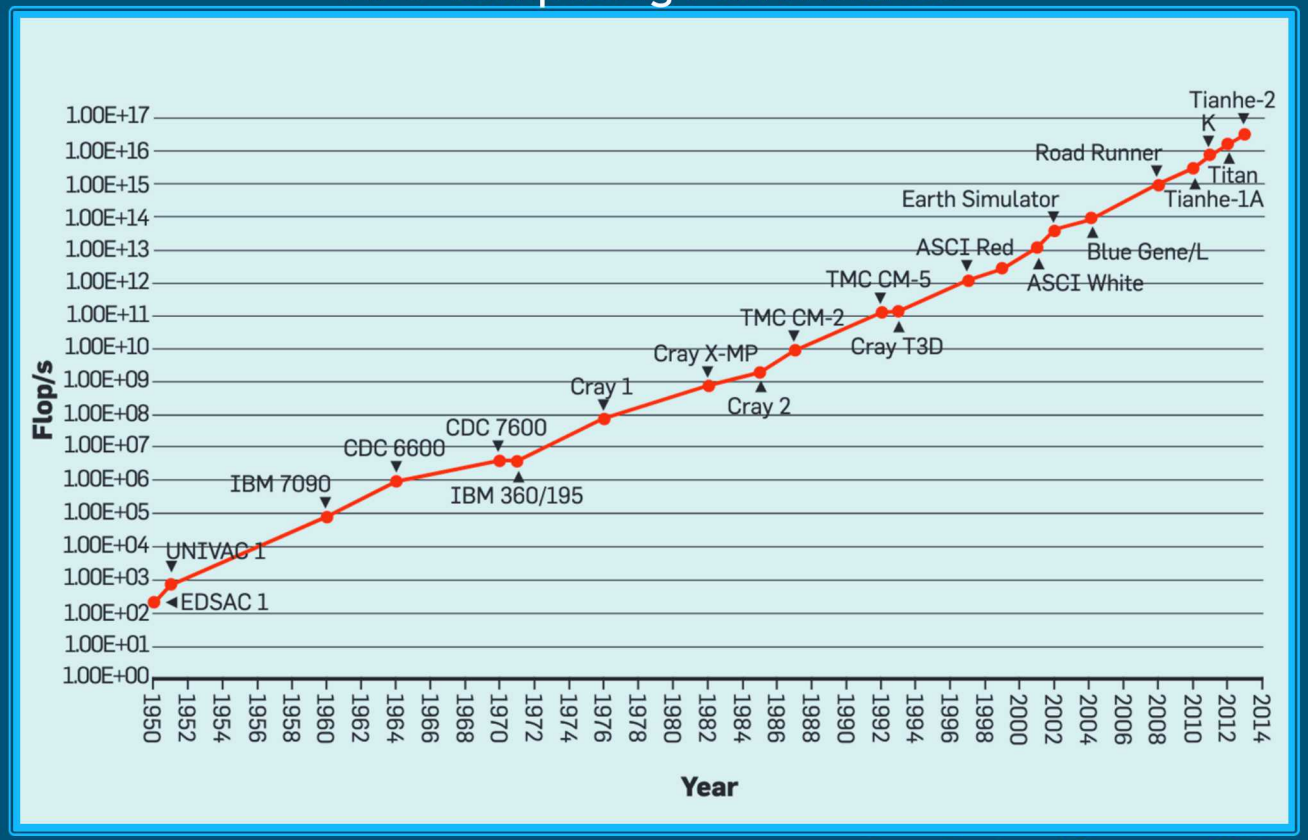
Predicting Severe Weather

Urban Science

High-Energy Physics, Astrophysics, Chemistry

Computer-Aided Design

HPC Computing Performance



From Reed and Dongarra, 2015

Scientific advancement is requiring ever-increasing compute power

Luckily, enabling algorithms are the same as always:

Dense Linear Algebra

Sparse Linear Algebra

Computation on Grids

Unstructured Grids

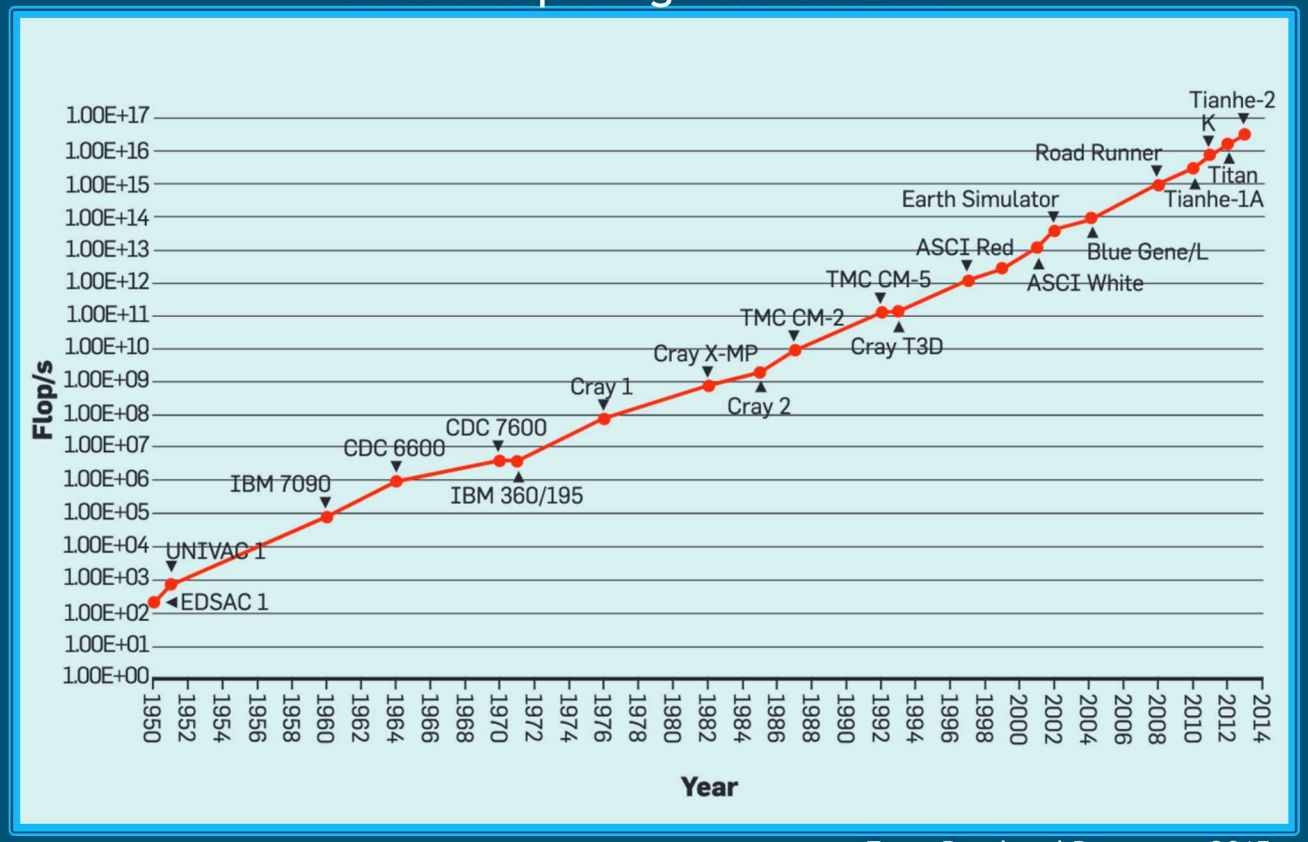
Spectral Methods

Particle Methods

Monte Carlo

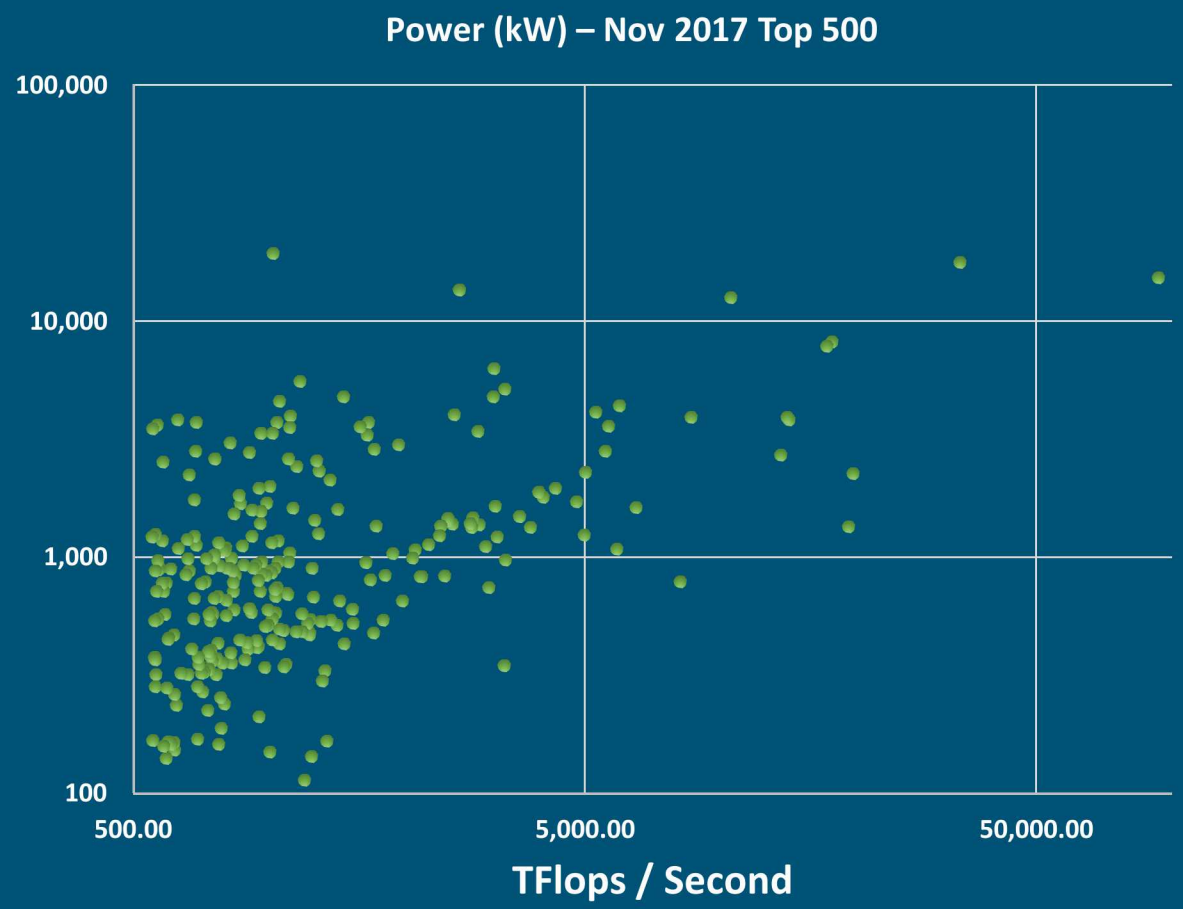
(Graph Analytics)

HPC Computing Performance



From Reed and Dongarra, 2015

So what's stopping us?



Supercomputers are increasingly limited by power consumption.

Exascale systems forecasted to be ~100MW.

That's more than 1/10th the total solar power generated in Colorado.

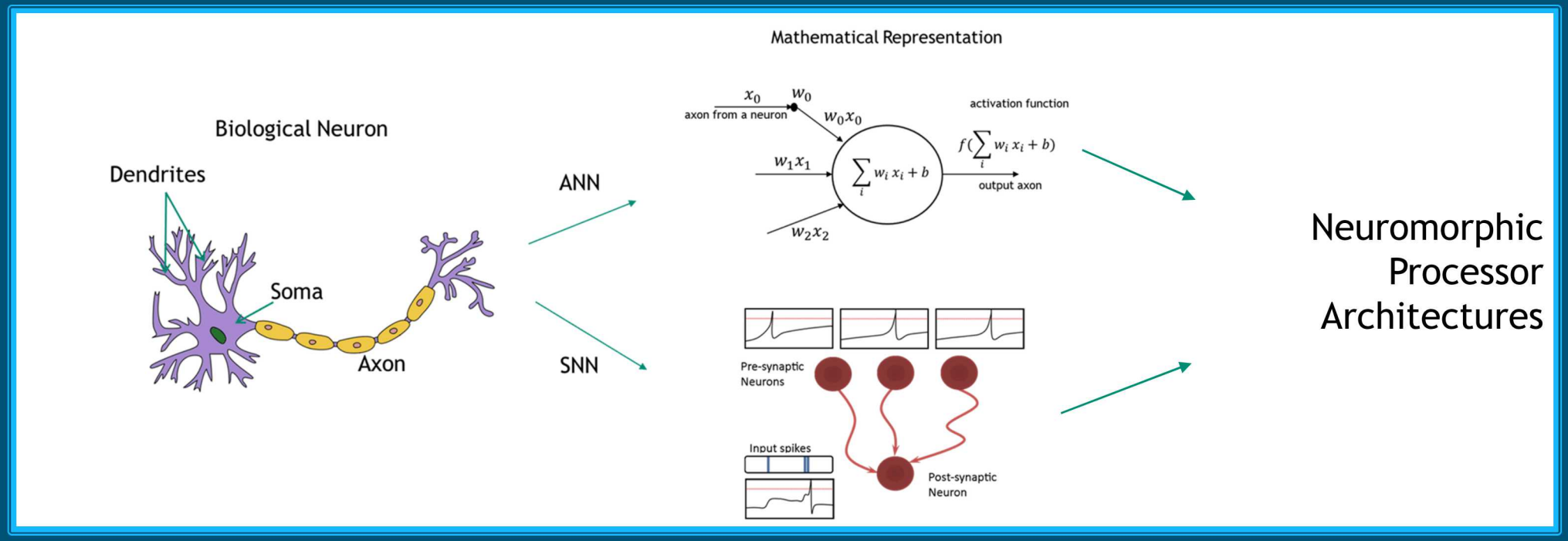
Or about 80,000 US homes.

Challenge: Can low-power neural-inspired or neuromorphic hardware solve the same problems for which we currently require traditional high-performance computing?

What is Neural-Inspired Computing?

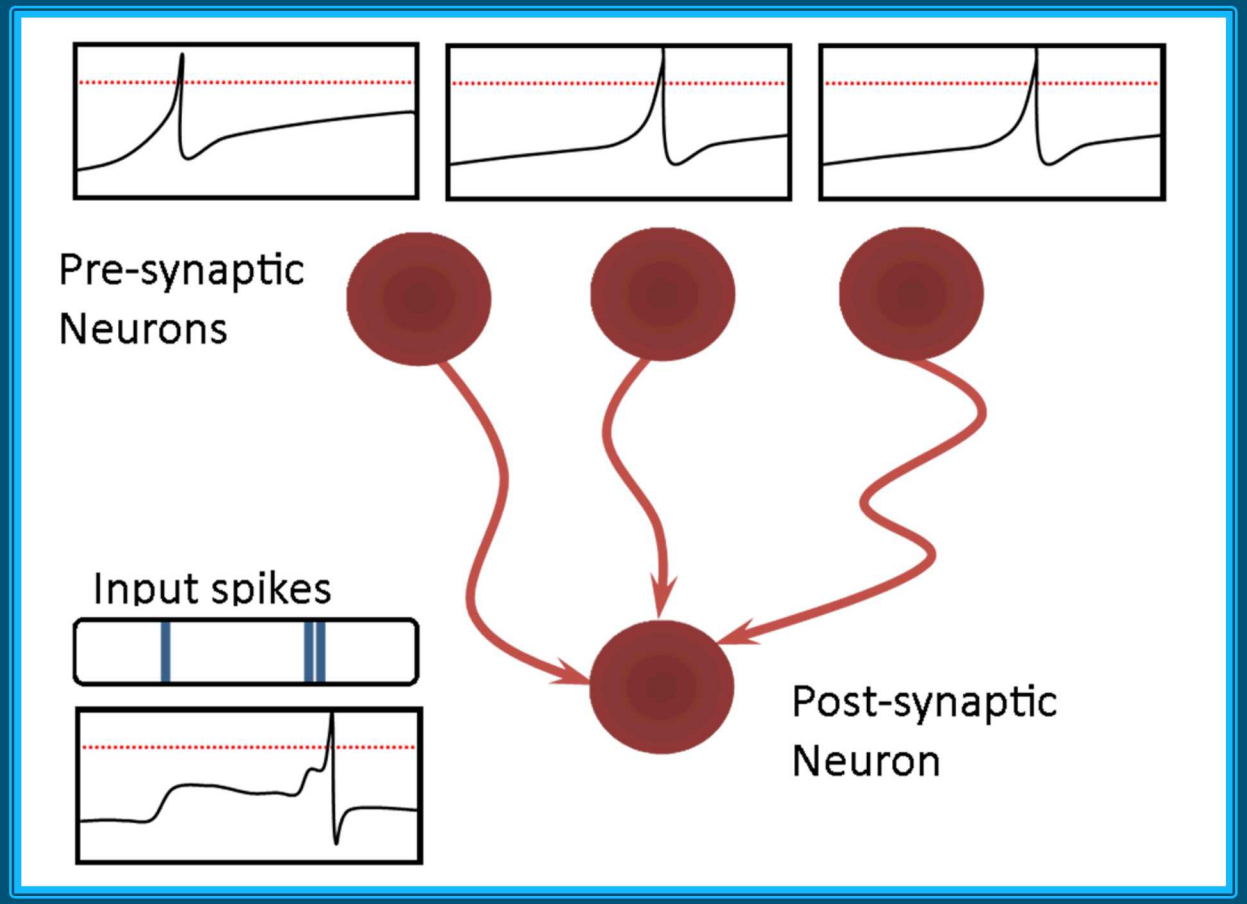
What is neural-inspired, neuromorphic, brain-inspired computing?

- Many terms
- Fundamental notion of taking inspiration from how the brain performs computation
- Evidence of 10-10,000x improvement in Energy-Delay Product (EDP)



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



Neuromorphic Processors: Where are we today?

Analog

- Focus on Kirchhoff Law – enabled computation
  - Neurons sum current across weighted synapses
  - Neural nodes sum current over weighted memristors
- Substantial energy and time savings
  - Non-trivial costs of precision
  - Practical issues limit size and integration with digital logic
- Ideal scenario
  - Train weights in situ
  - Compatible with linear algorithms

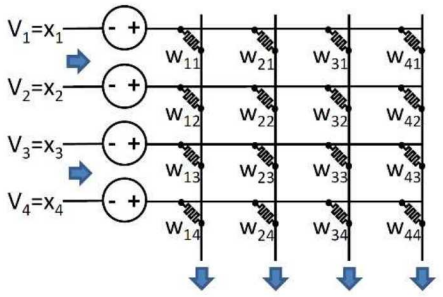
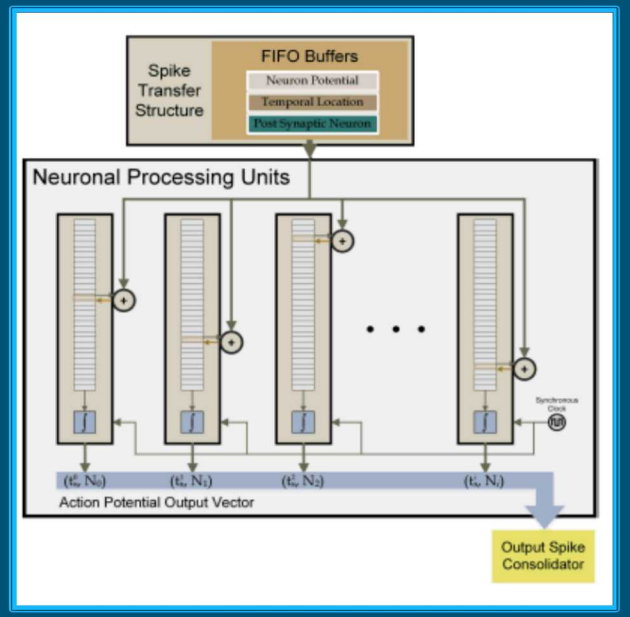


Fig 1: Analog RRAMs can be used to reduce the energy of a vector matrix multiply. The conductance of each RRAM represents a weight. Analog input values are represented by the input voltages or input pulse lengths. This allows all the read operations, multiplication operations and sum operations to occur in a single step. A conventional architecture must perform these operations sequentially for each weight resulting in a higher energy and delay.

Agarwal et al., E3S 2015

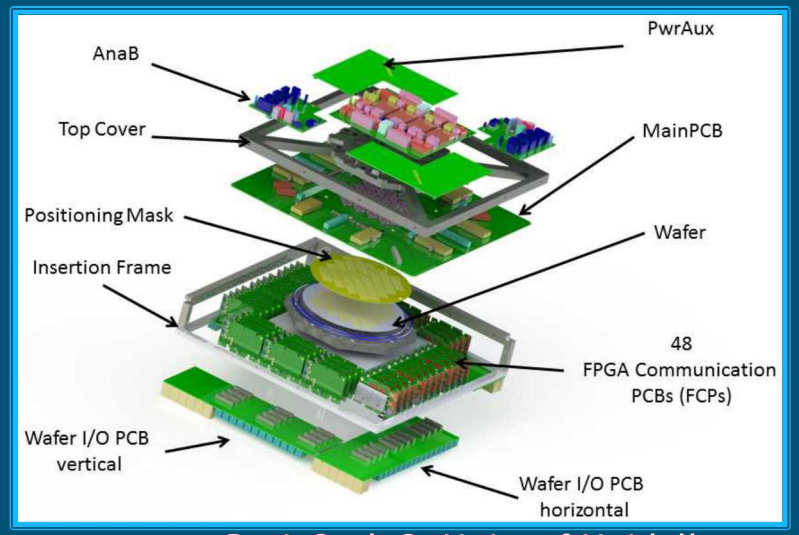
Digital

- Rely on event-driven "spiking" for communication
  - Communication only needed for '1's', not otherwise
  - Equivalent to large threshold gate networks + time dimension
- Substantial energy savings
  - Information in time dimension; limiting time savings
- Compatible and scalable using conventional technology
- Ideal scenario
  - Algorithms can be reframed in discrete spiking form
  - Learning algorithms are reformulated for spiking approaches

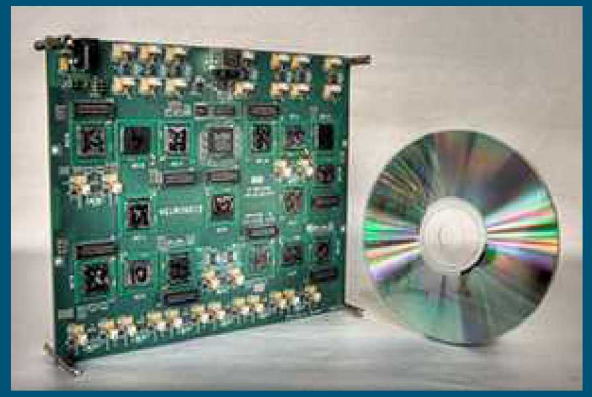


Neuromorphic Processors: Where are we today?

Analog

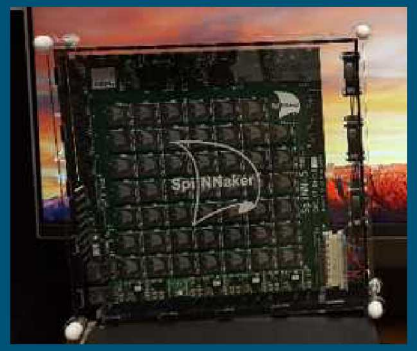


BrainScaleS, Univ. of Heidelberg
Photo: HBP Neuro. Comp. Platform

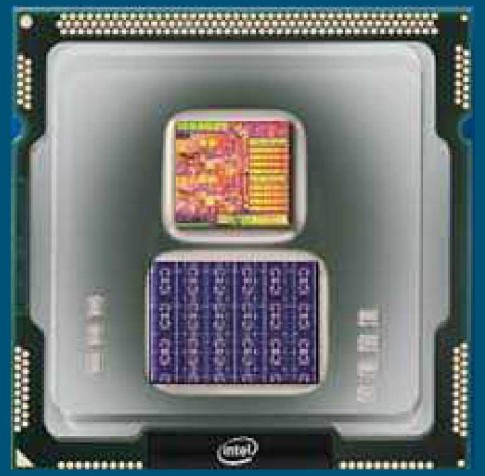


Neurogrid, Stanford Univ.
Photo: standford.edu

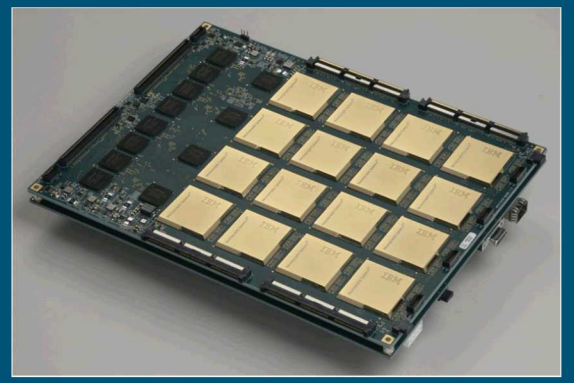
Digital



SpiNNaker, Univ. of Manchester



Intel Loihi
Photo: intel.com



IBM TrueNorth
Photo: IBM Corp

Spiking Neural Networks – Neuron Dynamics

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

For 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:

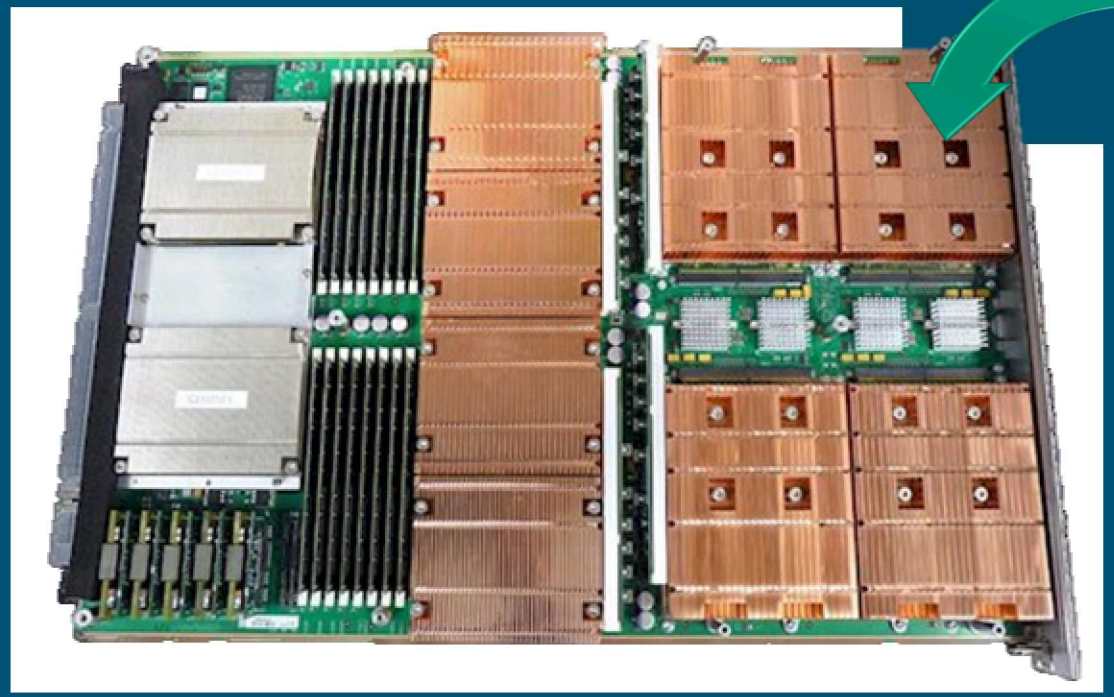
$$\widehat{V}_i(t+1) = V_i(t) + \sum_j w_{i,j} x_j(t - d_{i,j} + 1)$$

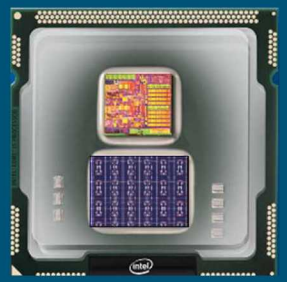
$$x_i(t+1) = \begin{cases} 1, \widehat{V}_i(t+1) > V_i^* \; and \; \eta_{i,t} < P_i \\ 0, \text{otherwise} \end{cases}$$

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

Each neuron processes these functions at **every time step** in **perfect parallel**.
So all you need to do is take your algorithm and formulate it
as a network of these independent processes.
Remember: Neurons are cheap. Spikes are cheaper.

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

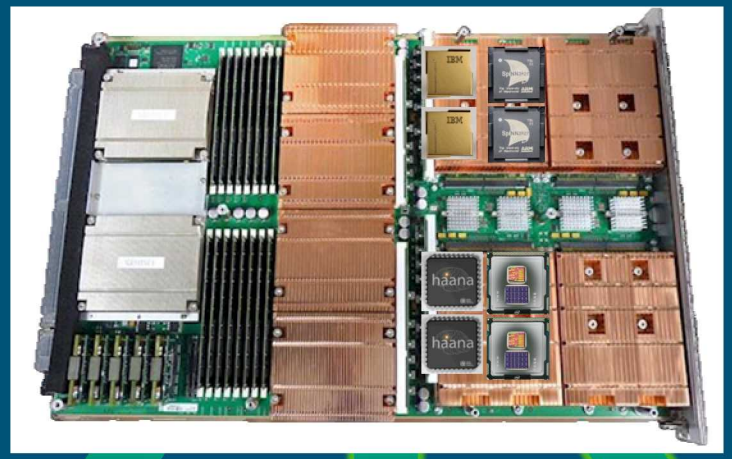


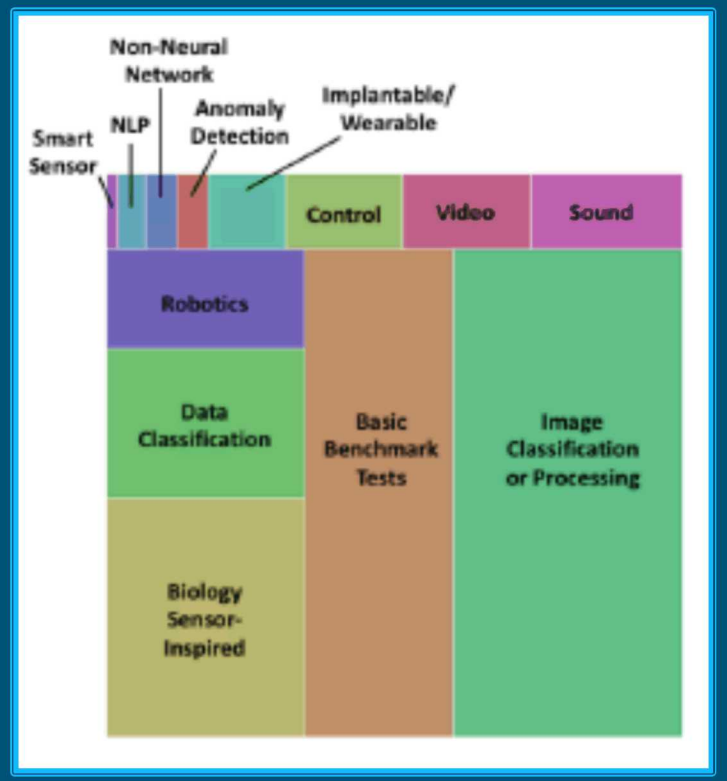


Emerging neuromorphic chips

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

Machine Learning, Yes… but neural platforms on HPC do not need to be limited to machine learning





Katie Schuman, ORNL, 2017

Biological-inspired neural algorithms

Machine Learning Deep Learning

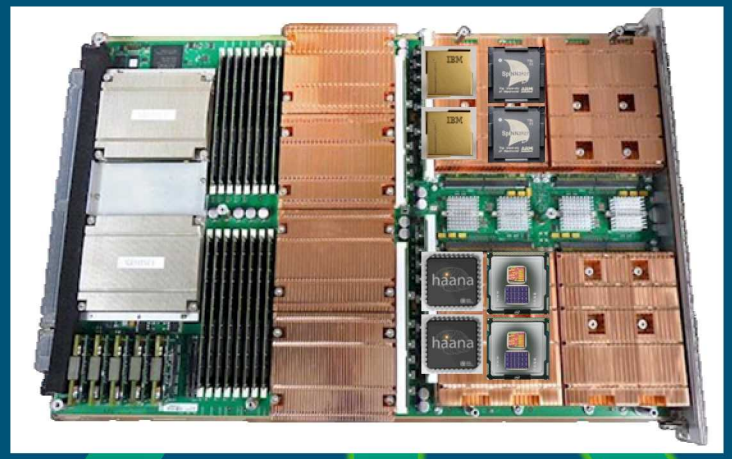
Surrogate Models; Reduced Order Modeling

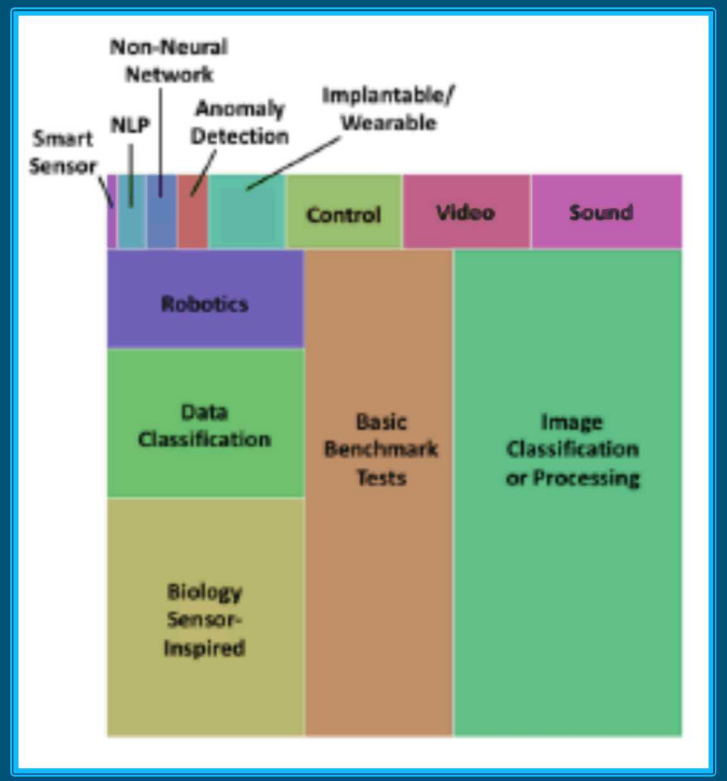
Neural-implemented numerical and scientific computing

Random Walk Methods

Graph Analytics

Machine Learning, Yes… but neural platforms on HPC do not need to be limited to machine learning





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

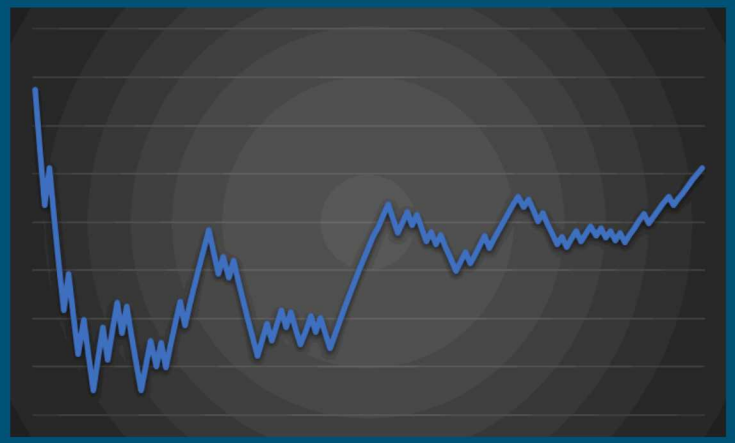
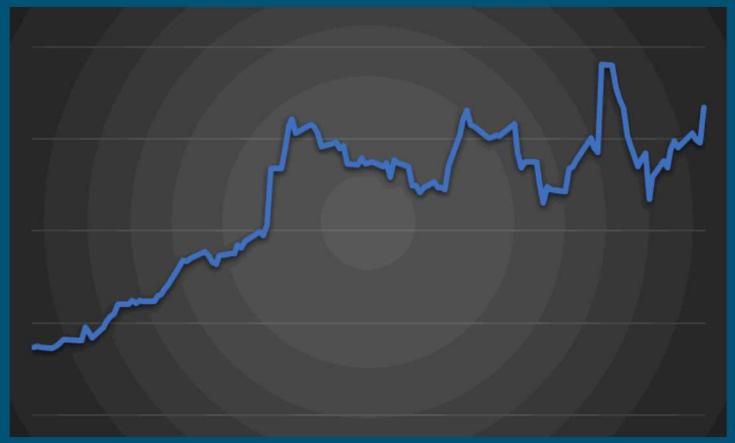
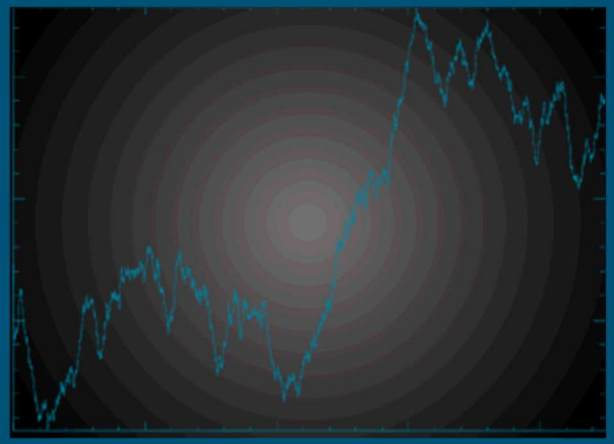
Model

Diffusion can be modeled either as a deterministic PDE or a stochastic process

- PDE solves the following equation

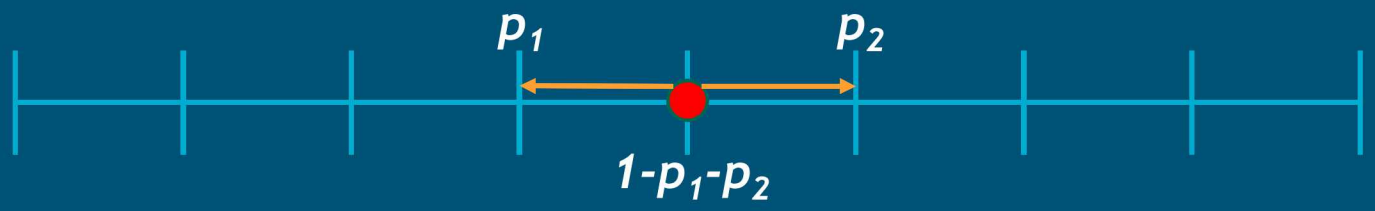
$$\frac{\partial C(x,t)}{\partial t} = D\frac{\partial^2 C(x,t)}{\partial x^2}$$

- Stochastic process implements many random walkers to statistically approximate a solution
  - Mean position of *N* walkers approaches expected mean of deterministic solution at rate of 1/sqrt(N)

One dimensional random walk case

Model:

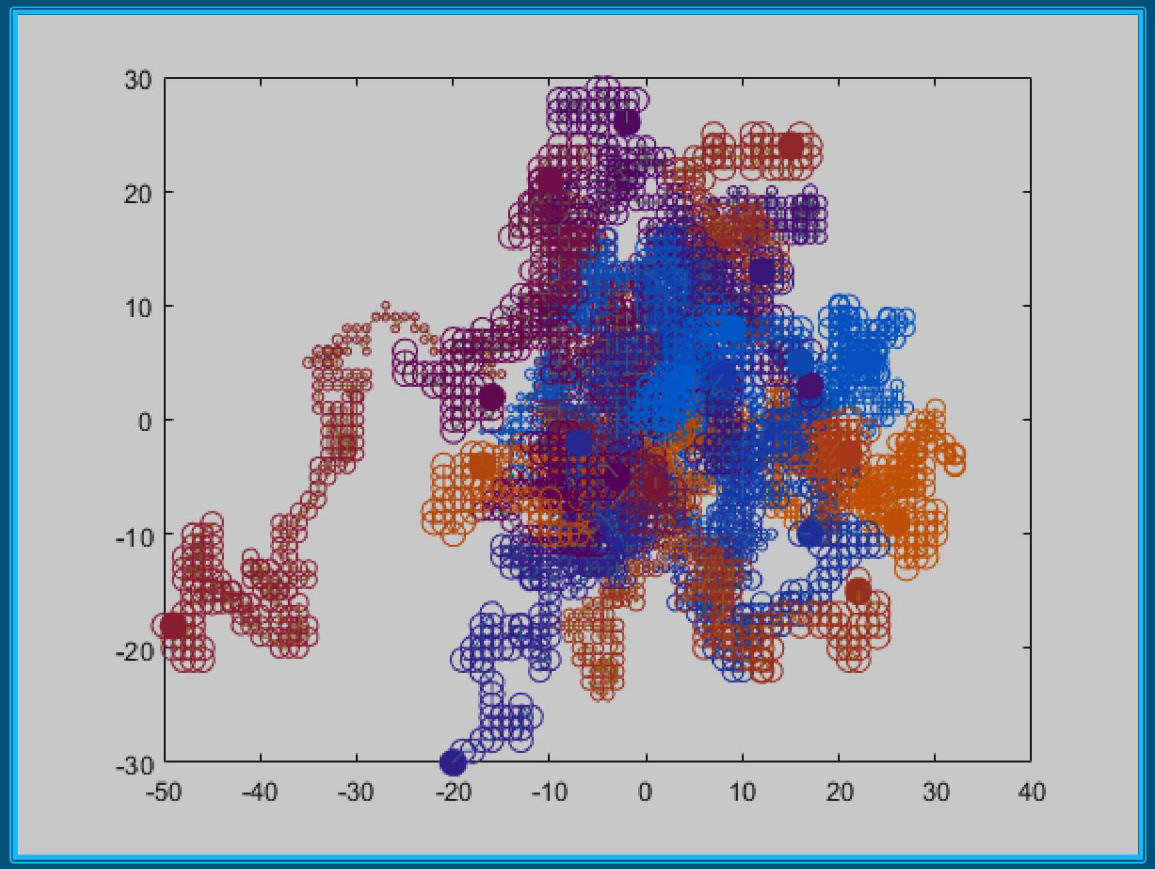


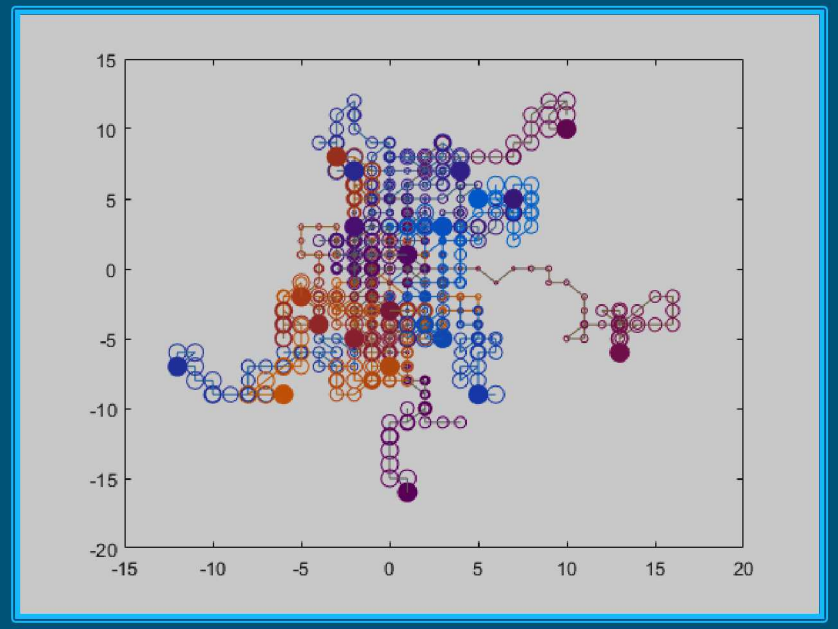
- Stochastic Brownian motion
- Particle can either move or not (1-D case: probability p_1 right or p_2 left, with $1-p_1-p_2$ for no move)
- Approximating PDE solutions requires sampling over MANY particles

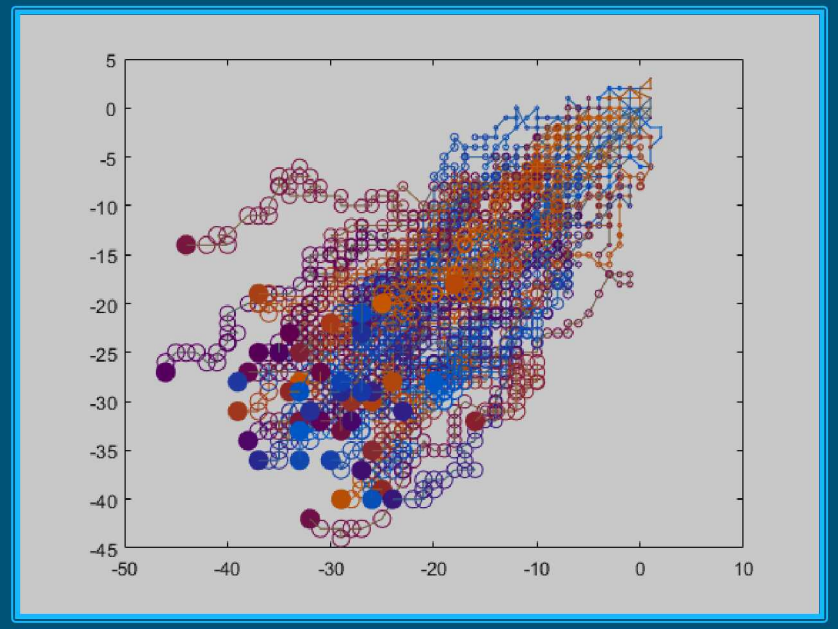
Goal: Ensemble of neurons that represent stochastic particles, such that

- Efficient to update (randomly add / subtract value)
- Has sparse representation
- Requires few neurons
- Scalable across multiple dimensions / multiple particles

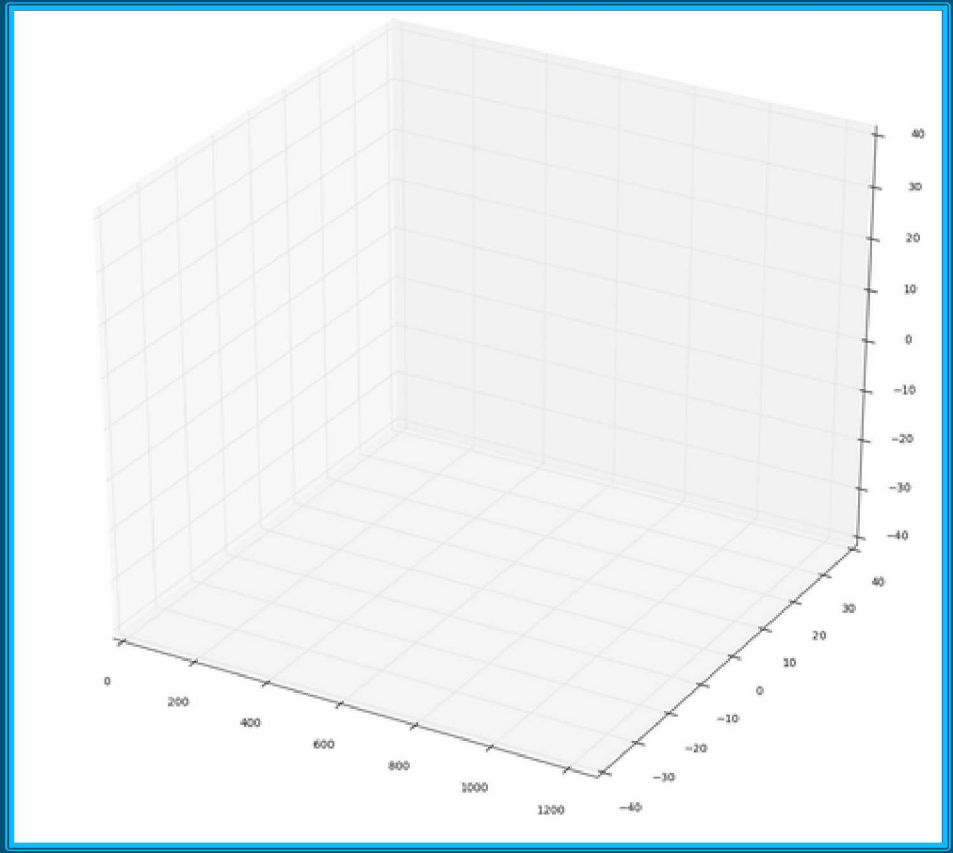
Neural random walkers evolve as expected

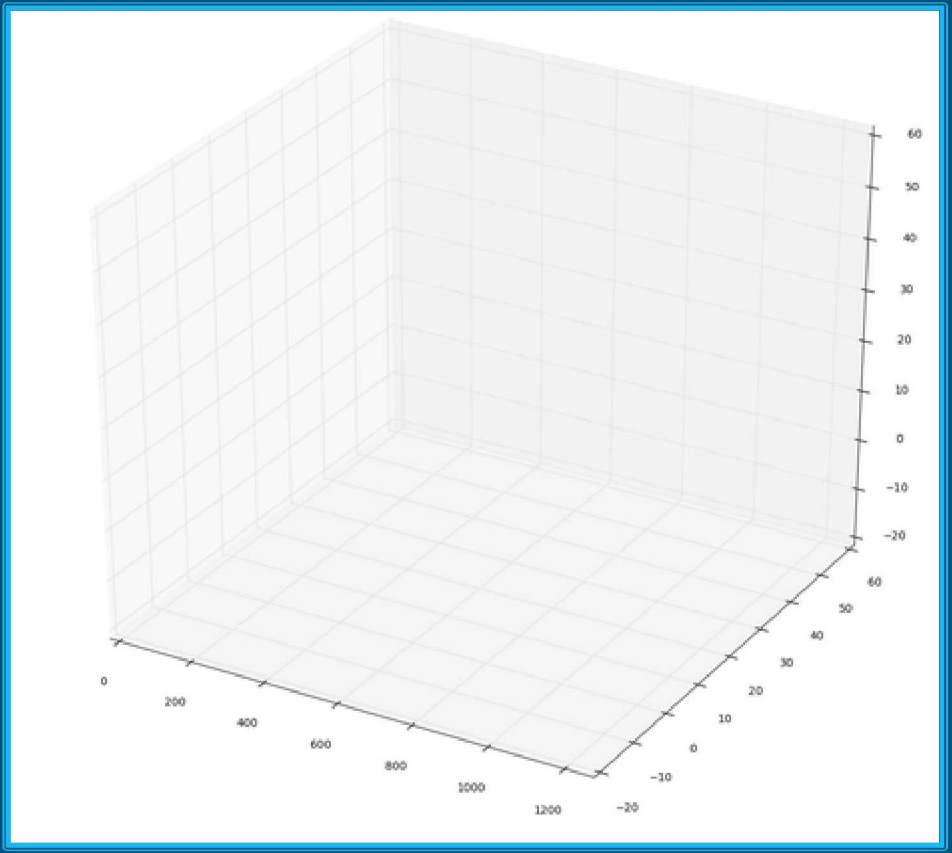




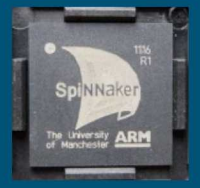


Neural random walkers evolve as expected



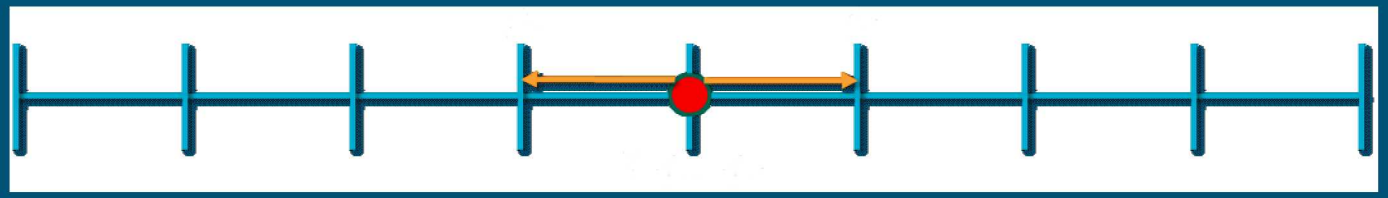


SpiNNaker 48 chip test board
250 walkers, 1250 time steps
with equal probability of moving (left) or strong bias (right)

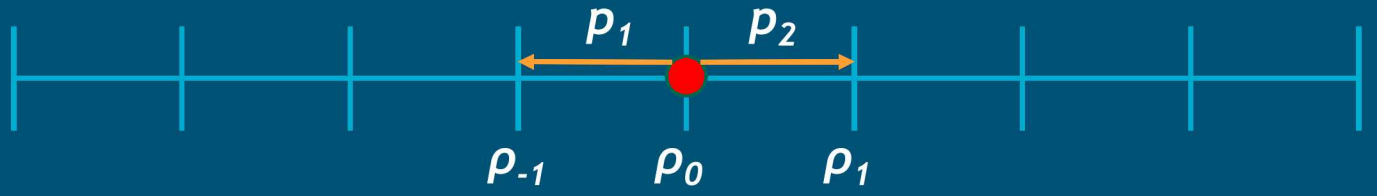


If we need lots of walkers, why are we using neurons to represent walkers? I heard that spikes would be cheaper…

Tracking *particles* over space → tracking *densities* at each location



Instead of each particle owning a population of neurons representing the particle's location…

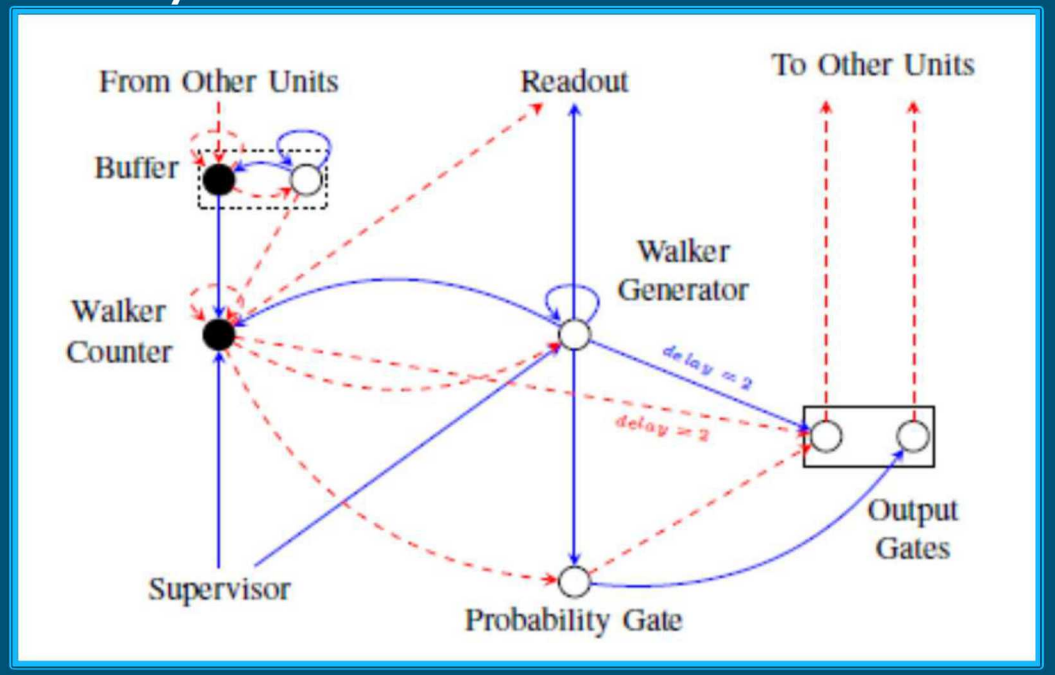


…have each location own a population of neurons that represent the probability density of particles

Density model repeats basic circuit at every vertex of mesh

Each vertex encodes density of particles in the internal potential of certain nodes

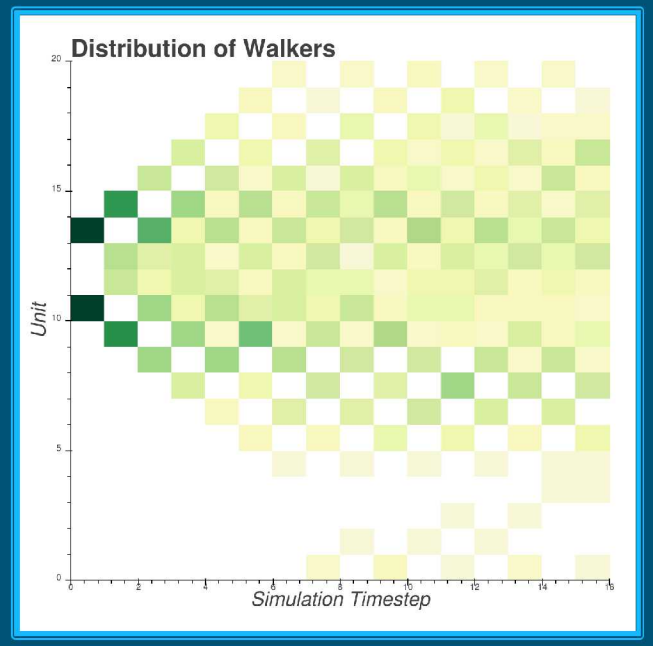
Each time step "hands off" particles to connected vertices according to probabilistic maps

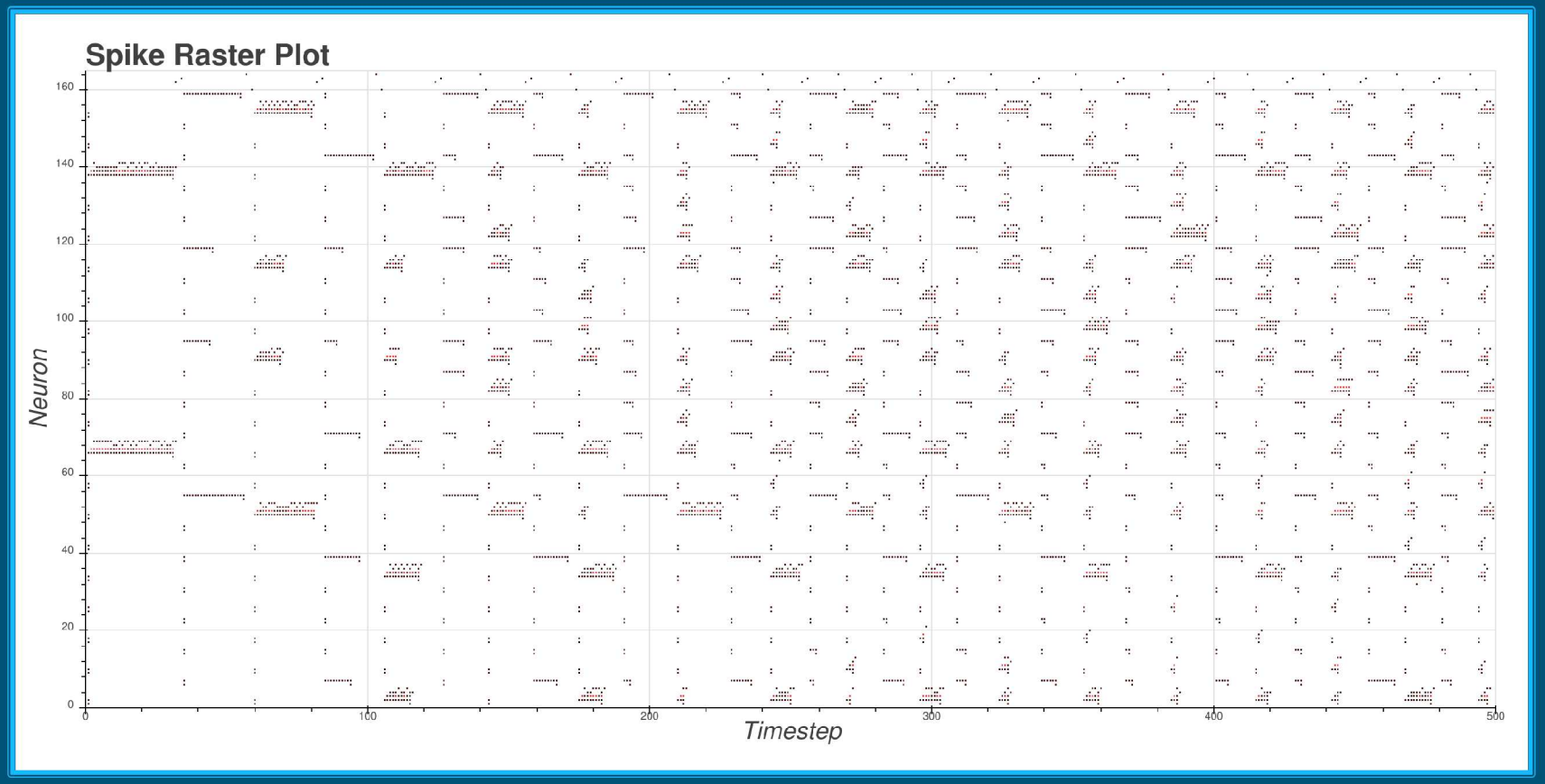


Measure	Cost (for k locations, simulating N walkers; 1-D case)
Walker memory	O(1)
Connection memory	O(k)
Total neurons	O(k)
Time per physical timestep	O(max(ρ_i)), where ρ_i is the density of walkers at each location
Position energy per timestep	O(N)
Update energy per timestep	O(N)

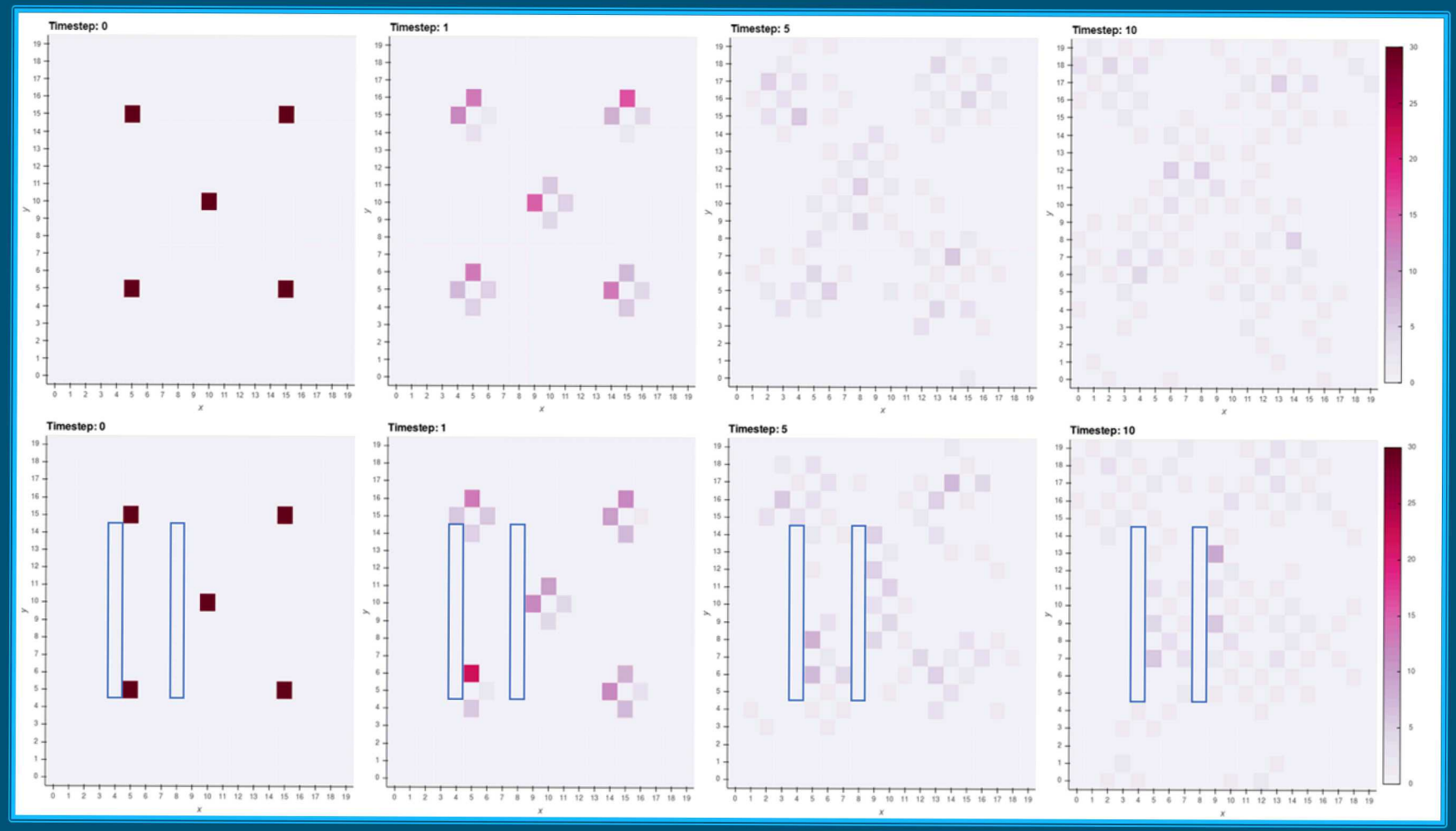
Severa, et al., IJCNN, 2018

Speed of simulation depends on density distribution of walkers

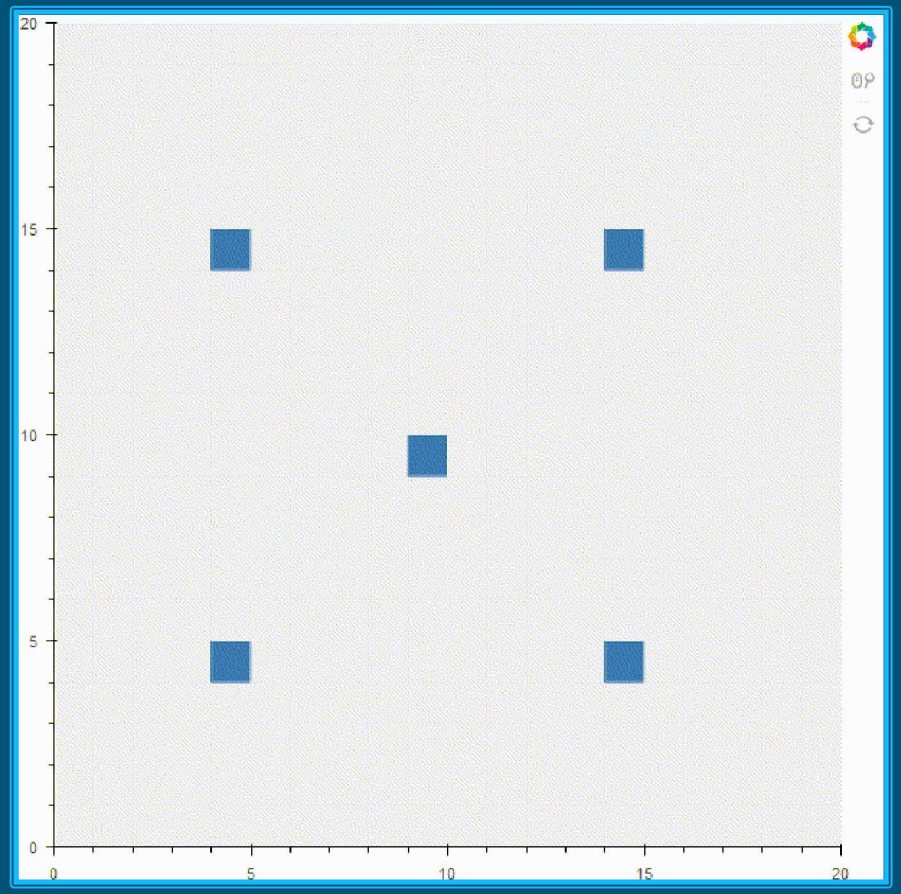




Density model is effective at looking at full distributions instantaneously



Crossbar Architectures Well-Suited for Local Connectivity



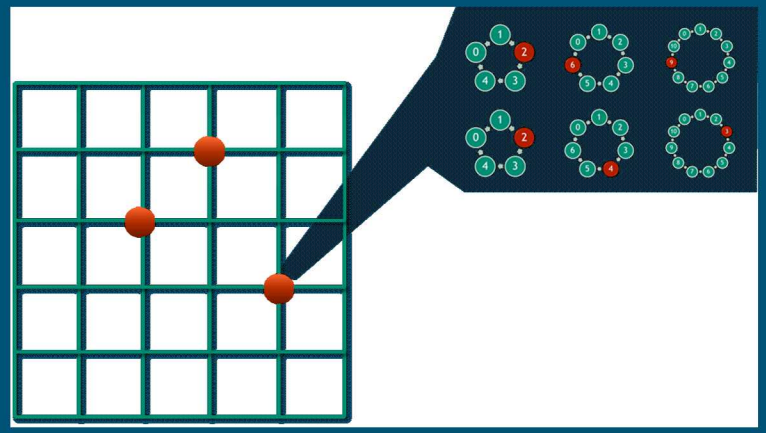
Simulation on IBM TrueNorth hardware (<100mW)

- 20x20 grid
- 30 walkers initialized at 5 points
- 2000 timepoints

Advantages of each method

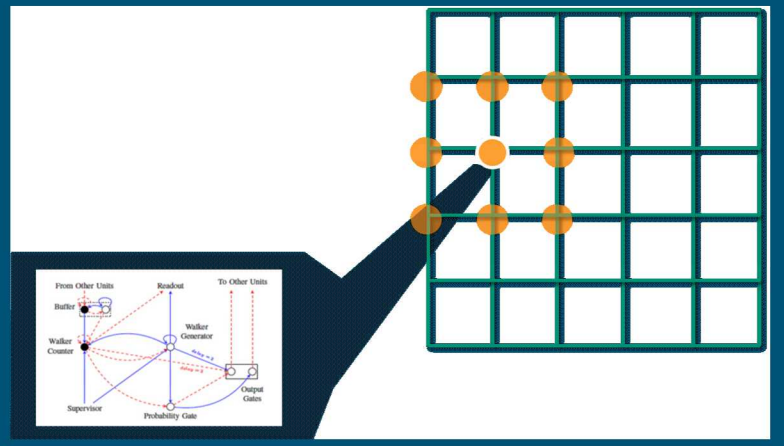
Particle method

- Path dependent behavior is readily available
- Until readout, communication is entirely local within particles (embarrassingly parallel)
- With unlimited neurons, can run in constant time



Density method

- Densities are readily available at all times
- Non-local or other complex graphs can easily be implemented
- With limited neurons, can tradeoff statistical approximation (i.e., number of walkers) with longer or shorter simulations



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 support of g.

A Class of Integro-PDEs with a Probabilistic Interpretation

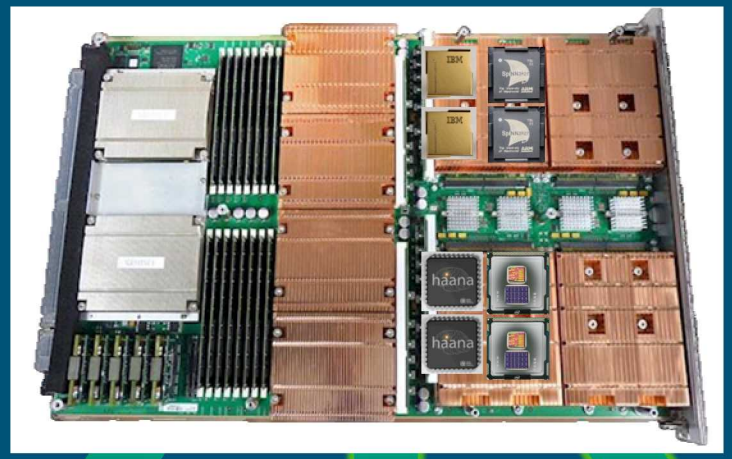
The IPDE-IVP

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

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

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	$\mathrm{d}X_t = v\mathrm{d}t$^

Machine Learning, Yes... but neural platforms on HPC do not need to be limited to machine learning



Biological-inspired neural algorithms

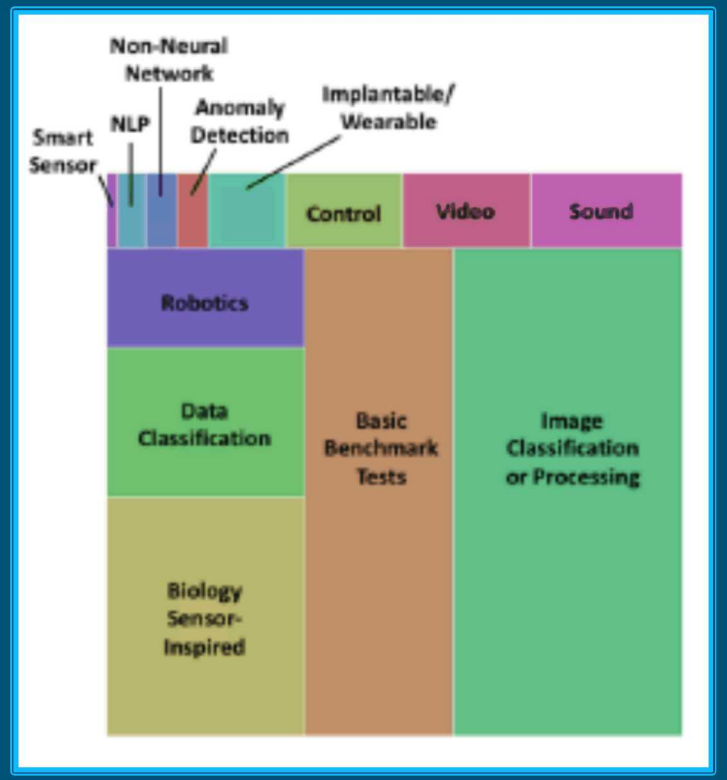
Machine Learning Deep Learning

Surrogate Models; Reduced Order Modeling

Neural-implemented numerical and scientific computing

Random Walk Methods

Graph Analytics



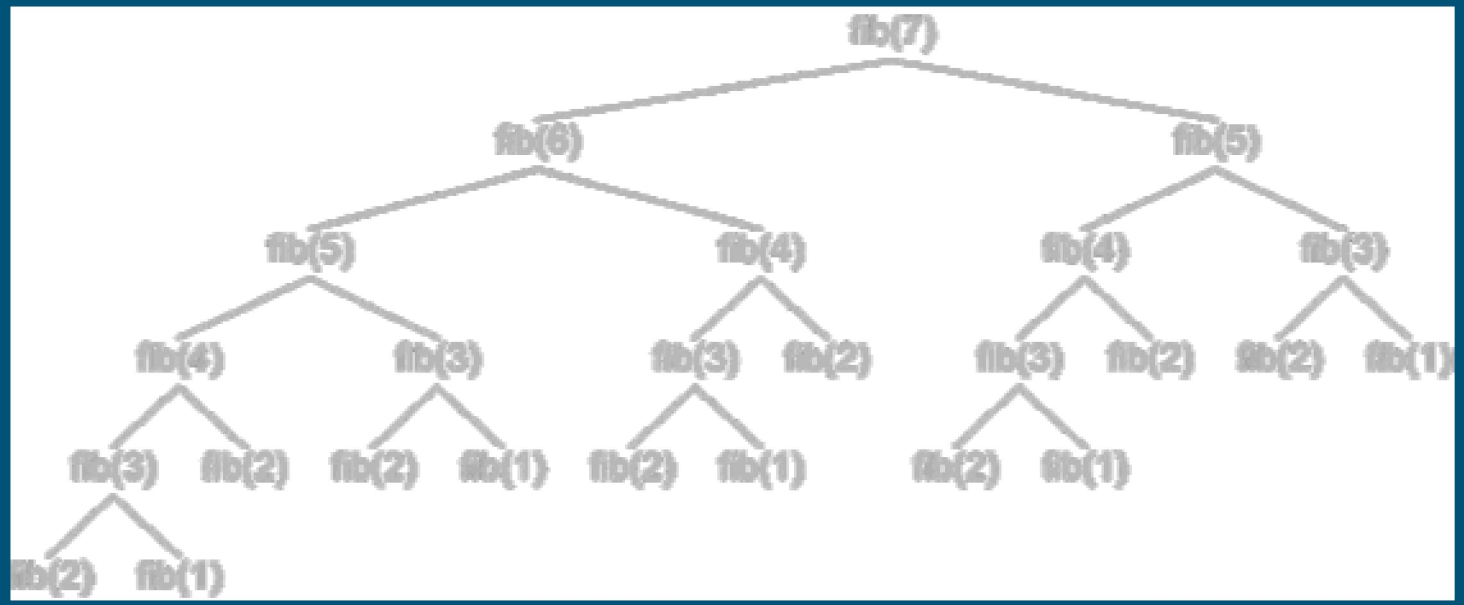
Katie Schuman, ORNL, 2017

Graph Algorithms

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

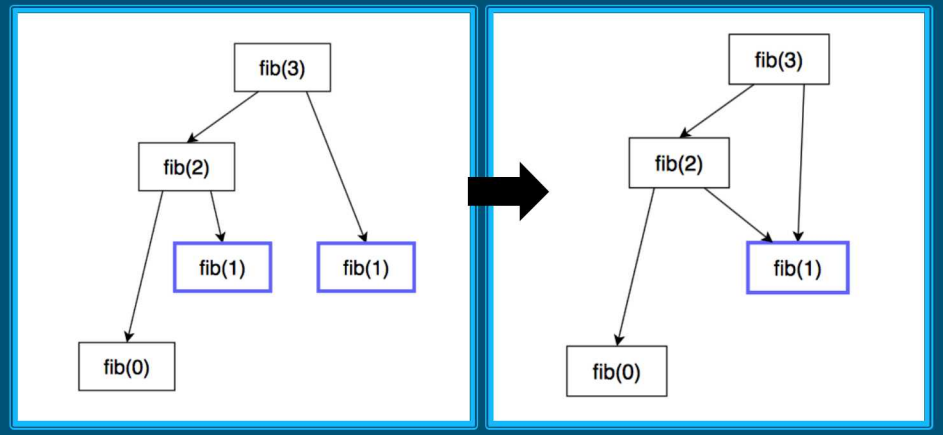
Dynamic programming consolidates redundant computation

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



Those who cannot remember the past are condemned to repeat it.

-Dynamic Programming



[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]

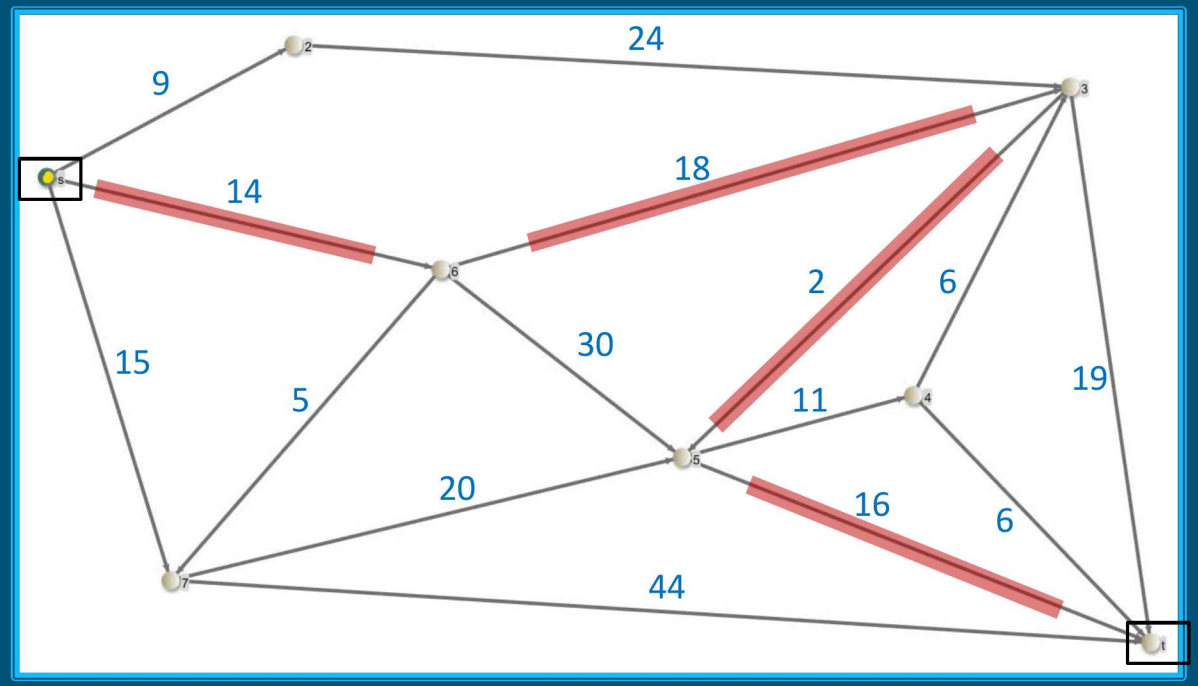
Graph Algorithms

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 (simulatable using recurrent neurons)
- **Novel temporal encoding:** time when neuron first fires represents value of dynamic program table entry

Aimone, et al., ICONS, 2019

Graph Algorithms

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

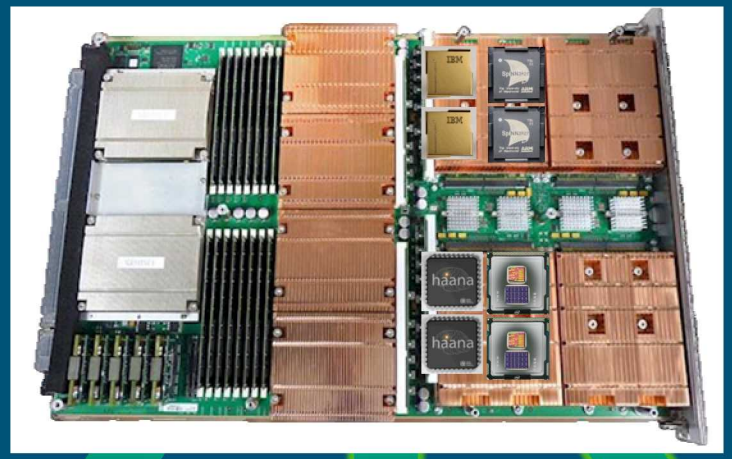
Wikipedia: 30 applications across diverse domains [https://en.wikipedia.org/wiki/Dynamic_programming]

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

Graph Algorithms – 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!

Machine Learning, Yes... but neural platforms on HPC do not need to be limited to machine learning



Biological-inspired neural algorithms

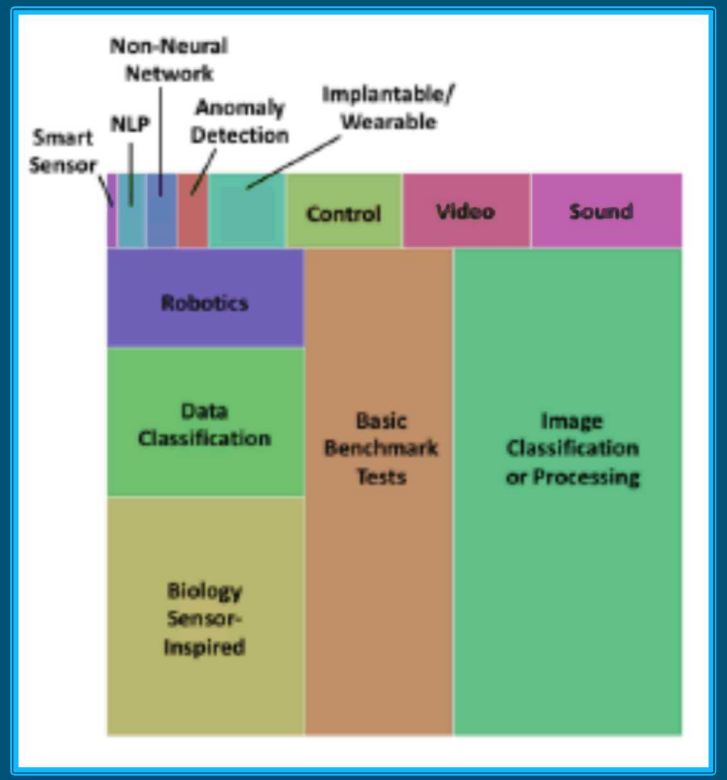
Machine Learning Deep Learning

Surrogate Models; Reduced Order Modeling

Neural-implemented numerical and scientific computing

Random Walk Methods

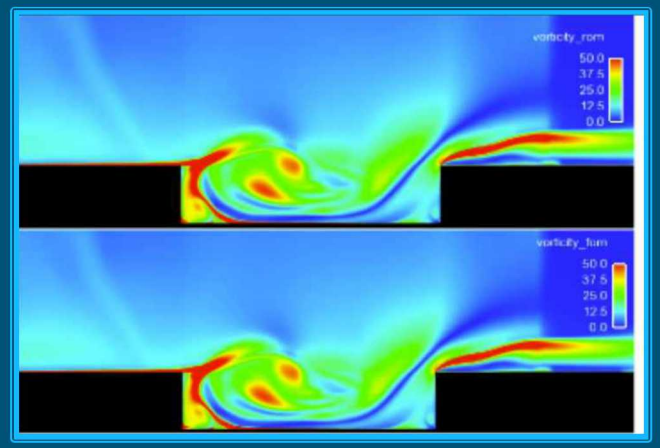
Graph Analytics



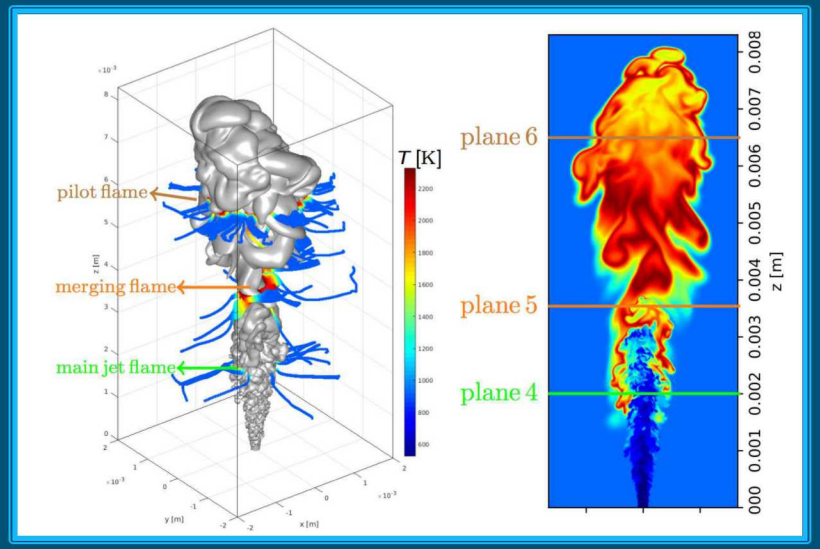
Katie Schuman, ORNL, 2017

Deep Learning for Reduced Order and Surrogate Models

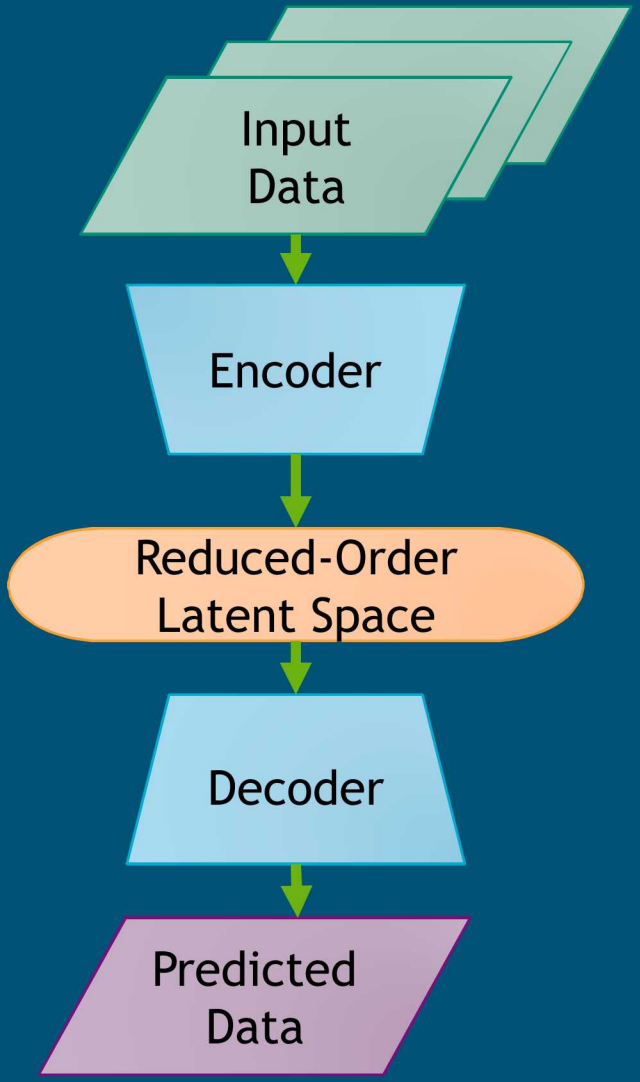
- High-Fidelity HPC physics simulations are expensive (e.g. petascale DNS may take days to run)
- For co-design problems or uncertainty quantification, many runs must be completed
- There exists a need for fast neural network approximations of physical systems
- Good news: Standard Deep Learning techniques work well for near-time predictions
  - Encoder-Decoder Convolutional Neural Networks
  - Iso-error and 100x faster than DNS



Turbulent Flow



DNS of n-dodecane Multi-injection Diesel Combustion



Images from Freno, Carlberg and Chen

Challenges for DL Reduced Order and Surrogate Models

Preventing Accumulating Errors

Despite low errors for near-term predictions, long-term predictions can exhibit growing errors

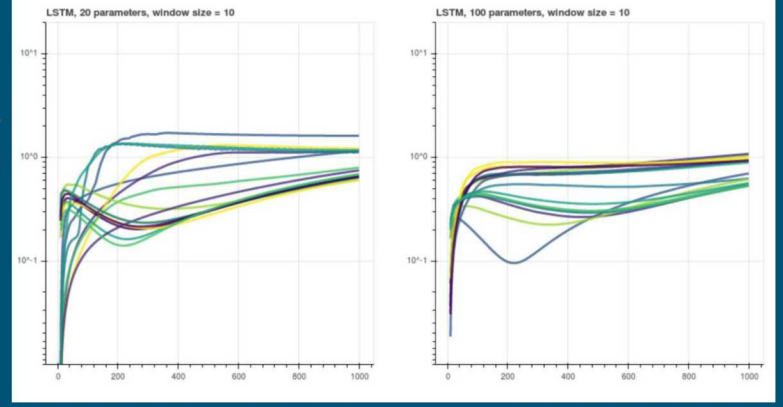
Shallow Ice-Sheet Model[1]

$$\tilde{h}_t(x,t,\boldsymbol{z}) = \nabla\cdot\left(\alpha\left(\frac{h^2(x,t)}{\beta(x,\boldsymbol{z})}+\Gamma H^{n+2}(x,t)|\nabla h(x,t)|^{n-1}\right)\nabla h(x,t)\right)+M(x,t)$$

$$h_t(x,t,\boldsymbol{z}) = \tilde{h}_t, \quad \text{if } h(x,t,\boldsymbol{z}) > b(x) \text{ or } \tilde{h}_t(x,t,\boldsymbol{z}) > 0,$$

$$h(x,t,\boldsymbol{z}) = 0, \quad \text{otherwise},$$

$$h(x,0) = \sqrt{\max\left(\frac{1}{2}-\frac{x^2}{900}, b^2(x)\right)}, \quad h(\pm L,t) = b(\pm L)$$



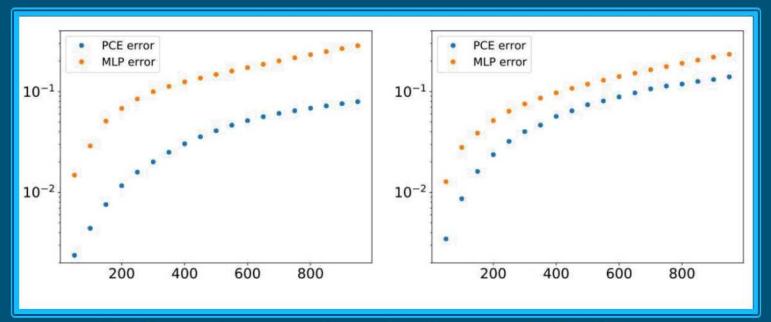
Low-Power Implementations

GPUs used for deep learning still consume 100+Watts. Neuromorphic systems have analog-readout issue.

Whetstone 'sharpening' trains neuromorphic encoder-decoder networks.[2]

Outperforming Existing Methods for QOI

Generally, the simulation is a necessary step in approximating a quantity-of-interest (QOI). For UQ efforts, traditional function approximation may out-perform QOI regression deep learning methods.



QOI estimate for Polynomial Chaos Expansion and Multi-Layer Perceptron[1]

[1]Perego, et al., SIAM Comp. Sci. Eng., 2019
[2]Severa, et al., Nature Machine Intelligence, 2019

Conclusions

- Neural Inspired and Neuromorphic approaches have a wide range of potential impacts in numerical computing.
- While performance (throughput) is key, energy is also a critical limiting factor
- Yes, Deep Learning has a place, but new computer architectures mean re-thinking all types of algorithms
- Thanks!

Images from SPARTA DSMC codes