Parallelization and runtime optimization

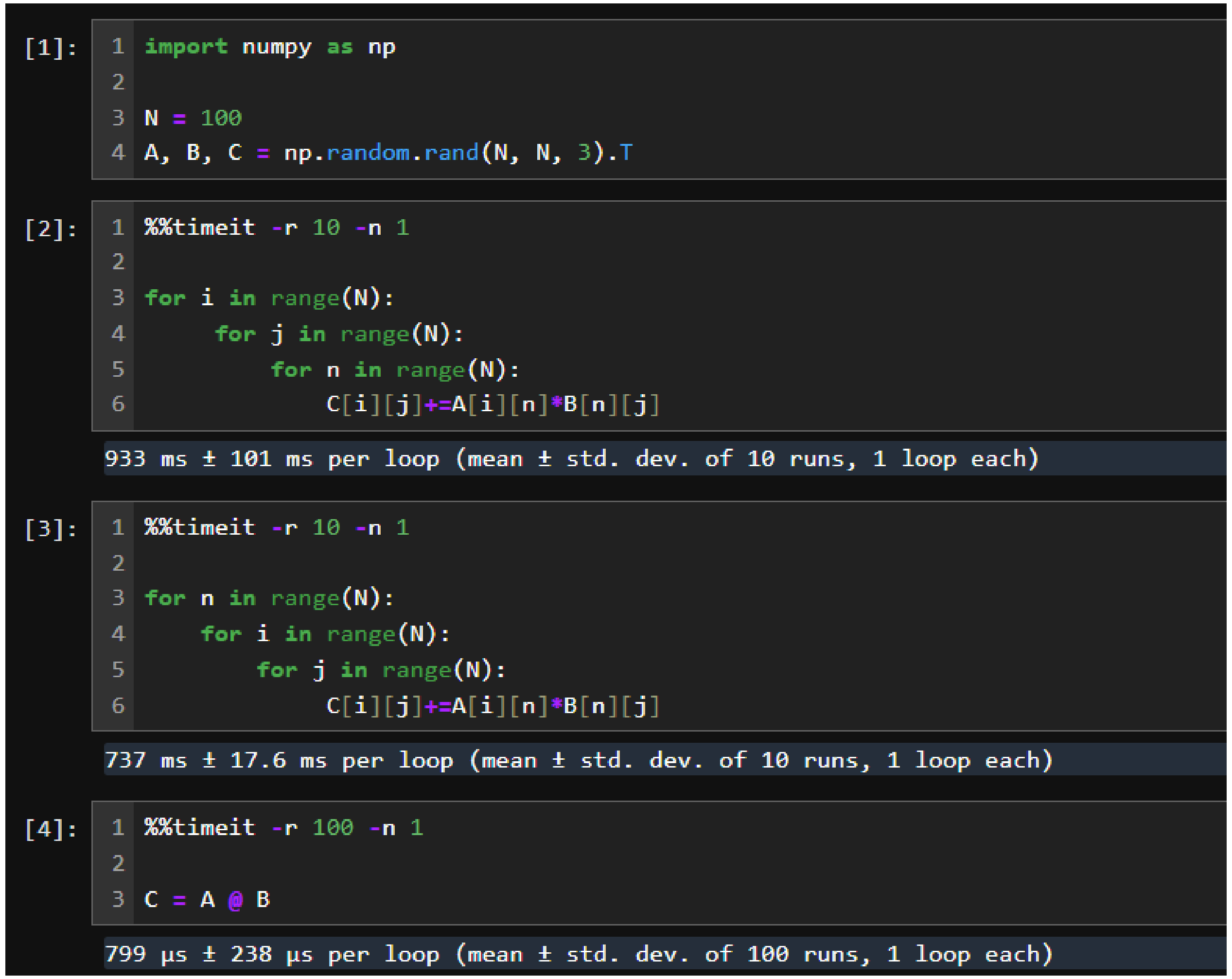
Algorithmic optimization and outsourcing

A great and simple example of algorithmic optimization is the naive matrix multiplication. *jupyter notebook for reference*

Matrix Multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$



[1]:
```
import numpy as np

N = 100
A, B, C = np.random.rand(N, N, 3).T
```

[2]:
```
%%timeit -r 10 -n 1

for i in range(N):
     for j in range(N):
         for n in range(N):
             C[i][j]+=A[i][n]*B[n][j]
```

```
933 ms ± 101 ms per loop (mean ± std. dev. of 10 runs, 1 loop each)
```

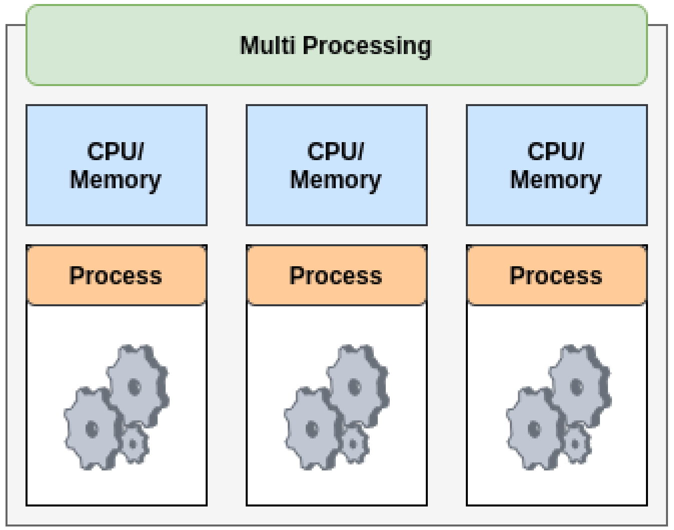
[3]:
```
%%timeit -r 10 -n 1

for n in range(N):
    for i in range(N):
        for j in range(N):
            C[i][j]+=A[i][n]*B[n][j]
```

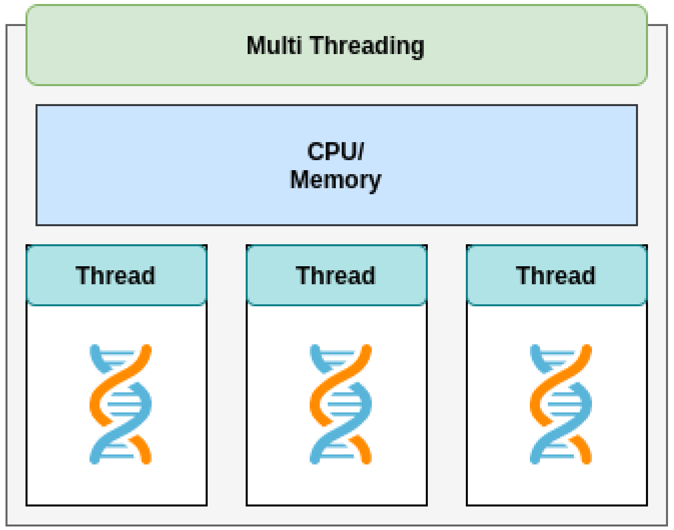
```
737 ms ± 17.6 ms per loop (mean ± std. dev. of 10 runs, 1 loop each)
```

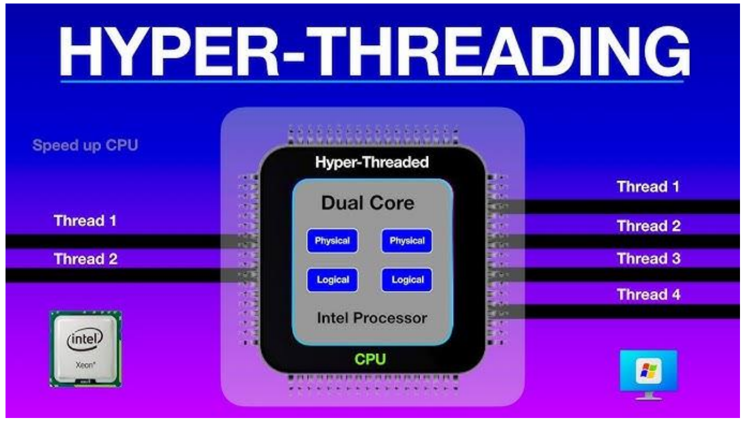
[4]:
```
%%timeit -r 100 -n 1

C = A @ B
```

```
799 µs ± 238 µs per loop (mean ± std. dev. of 100 runs, 1 loop each)
```

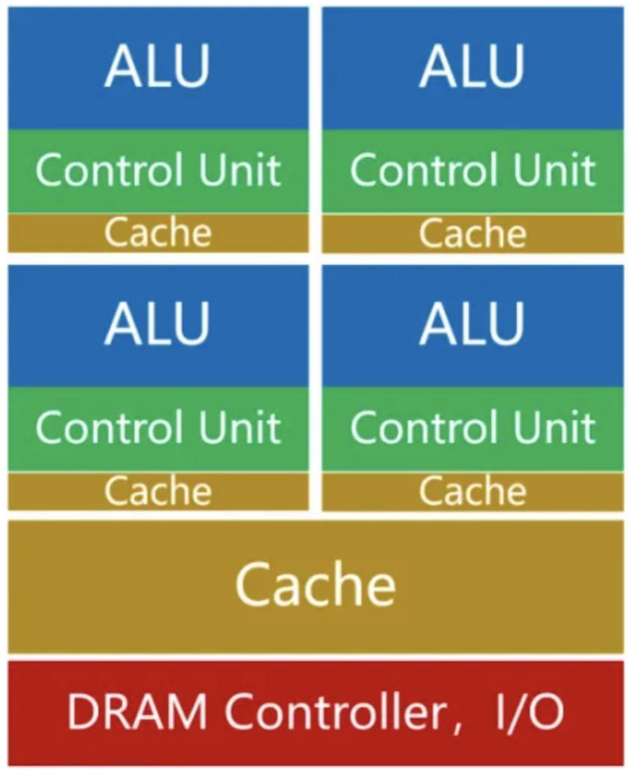
Three depth levels of parallelization



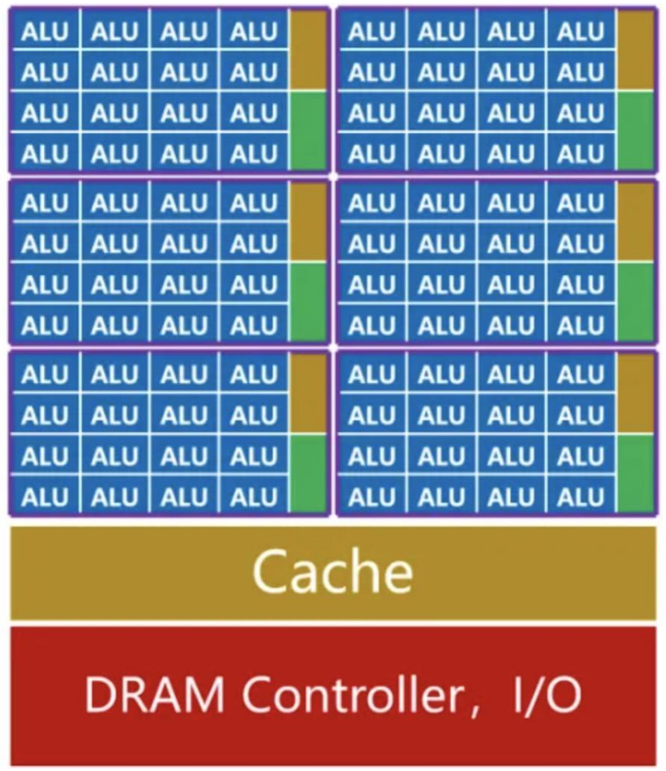




CPU vs GPU



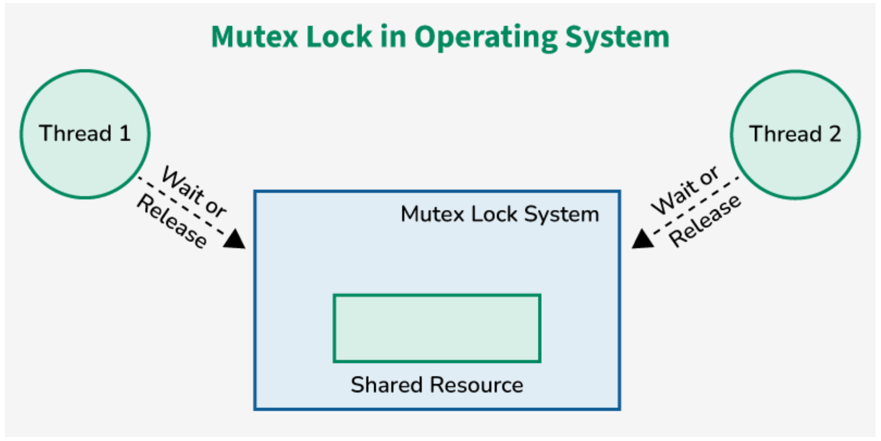
CPU

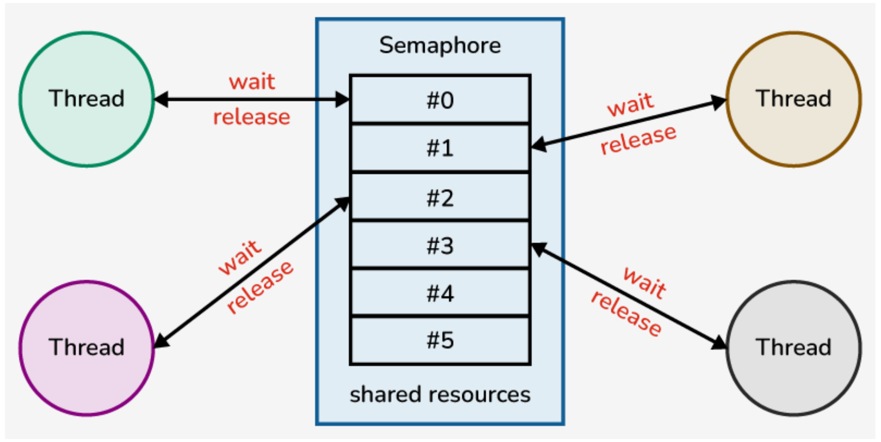


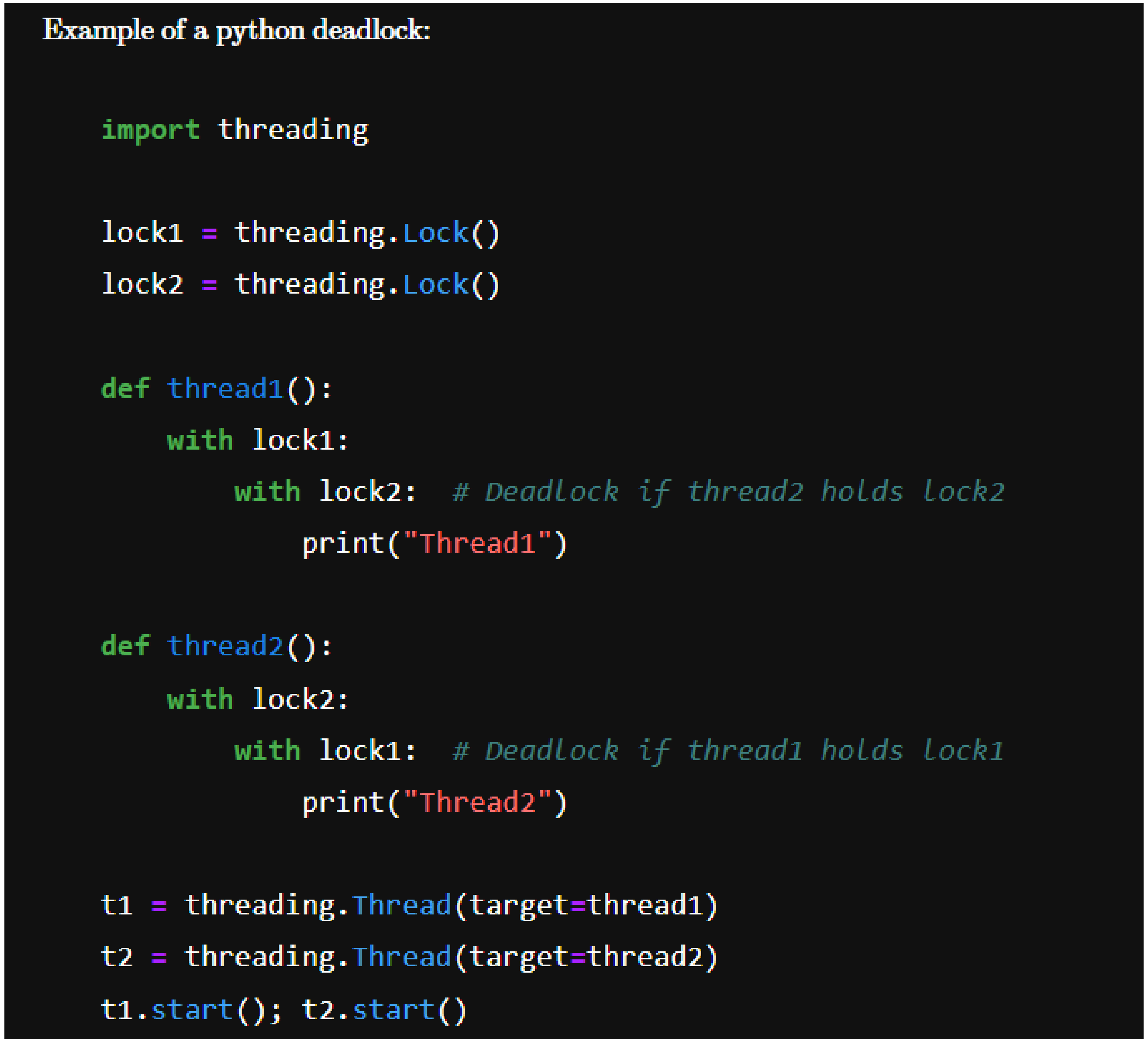
GPU

Core parallelization concepts

- **Daemon Processes/Threads -** Background processes that terminate when the main program exits
- **Locks/Mutexes**: Ensure only one thread accesses a resource at a time.
- **Semaphores-** Limit access to a resource to N threads.
- **Barriers-** Threads wait until all reach a certain point before proceeding.
- **Deadlocks-** two or more processes block forever, each holding a resource the other needs
- **Race Conditions-** conditions of memory (rights) access between processes/threads
- **Load Balancing**






Example of a python deadlock:

```
import threading

lock1 = threading.Lock()
lock2 = threading.Lock()

def thread1():
    with lock1:
        with lock2:  # Deadlock if thread2 holds lock2
            print("Thread1")

def thread2():
    with lock2:
        with lock1:  # Deadlock if thread1 holds lock1
            print("Thread2")

t1 = threading.Thread(target=thread1)
t2 = threading.Thread(target=thread2)
t1.start(); t2.start()
```

Parallelization frameworks

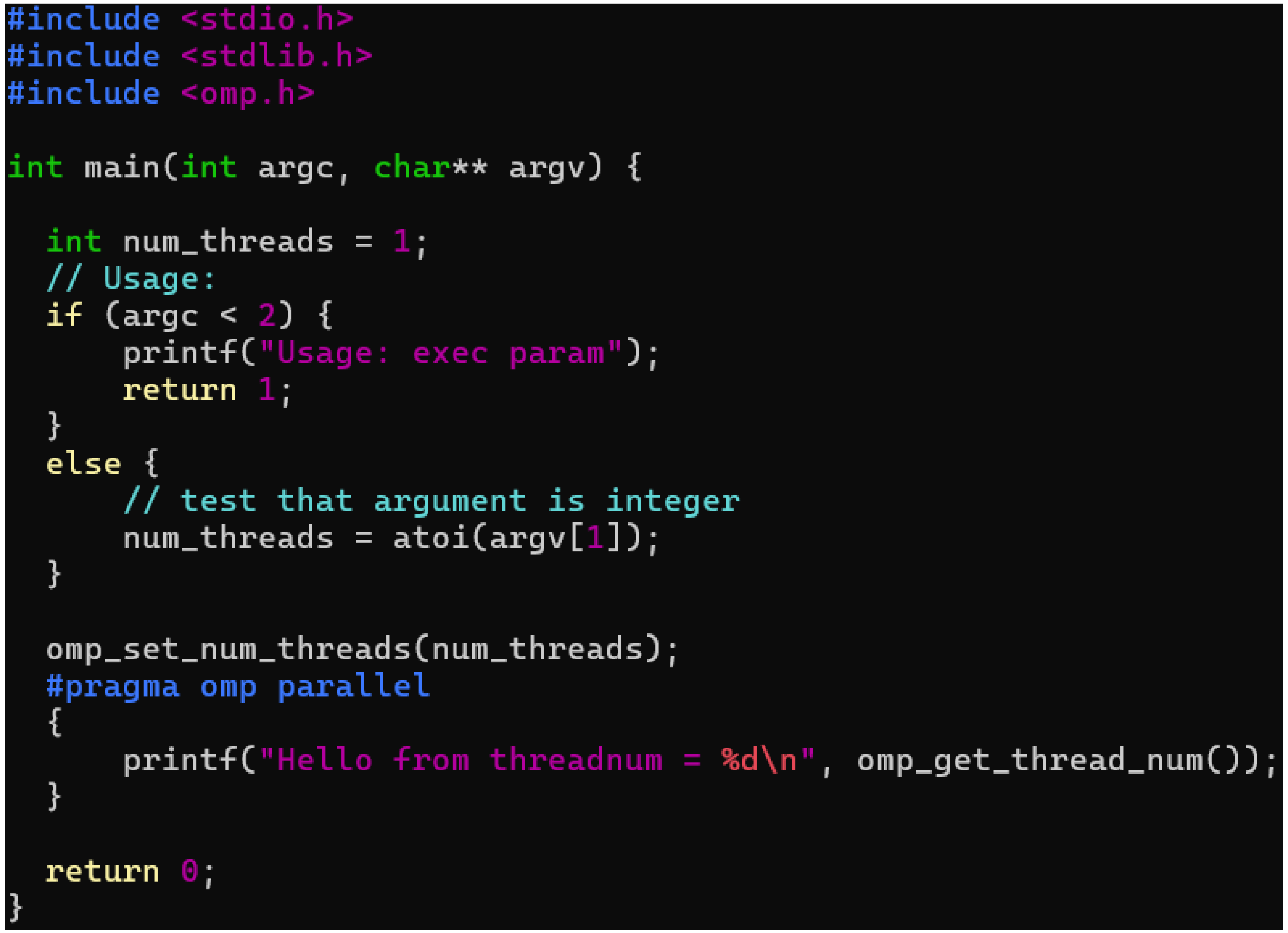
Multiprocessing

multiprocessing

threading

Time to look at some real code

Snippet that calls for each process
smooth transition to jupyter notebook



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char** argv) {

  int num_threads = 1;
  // Usage:
  if (argc < 2) {
      printf("Usage: exec param");
      return 1;
  }
  else {
      // test that argument is integer
      num_threads = atoi(argv[1]);
  }

  omp_set_num_threads(num_threads);
  #pragma omp parallel
  {
      printf("Hello from threadnum = %d\n", omp_get_thread_num());
  }

  return 0;
}
```

Neuromorphic Computing

Neuromorphic computing is a broad concepts that refers to anything that relates to the computation using either live neurons or architectures that mimic the neural structure of human brain. Today we'll take a look at the Loihi neuromorphic chip designed by Intel, its performance and use in video and signal processing tasks

Loihi architecture

Loihi novelty:

- Sparse network compression
- Core-to-core multicast
- Variable synaptic formats
- Population-based hierarchical connectivity

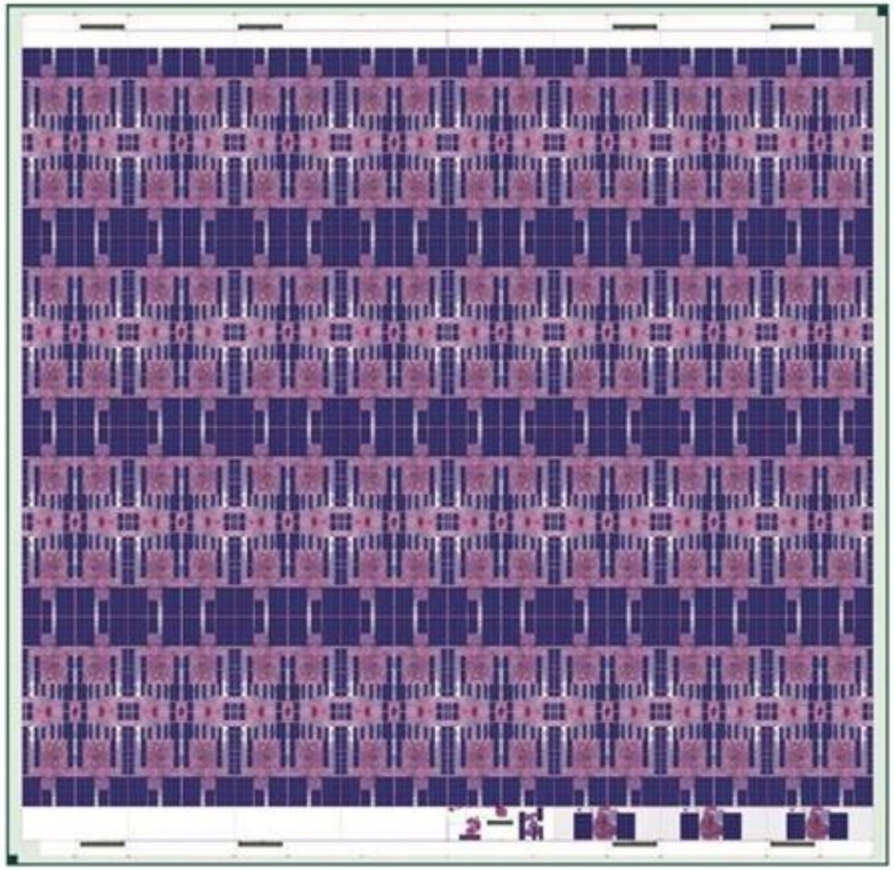


Figure 7. Loihi chip plot.

Mesh operation and synchronization

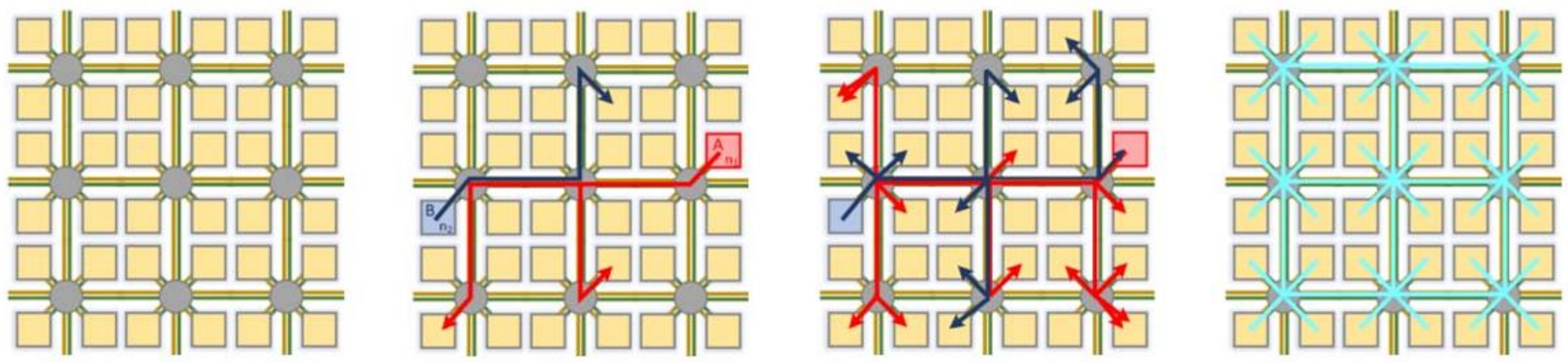


Figure 2. Mesh Operation: first box, initial idle state for time-step *t* (each square represents a core in the mesh containing multiple neurons); second box, neurons n_1 and n_2 in cores *A* and *B* fire and generate spike messages; third box, spikes from all other neurons firing on time-step *t* in cores *A* and *B* are distributed to their destination cores; and fourth box, each core advances its algorithmic time-step to $t + 1$ as it handshakes with its neighbors through barrier synchronization messages.

Constraints

Loihi architecture can support arbitrary multigraph networks subject to the cores' resource constraints:

- The total number of neurons assigned to any core may not exceed 1,024 (N_cx)
- The total synaptic fan-in state mapped to any core must not exceed 128 KB (N_syn × 64b, subject to compression and list alignment considerations)
- The total number of core-to-core fan-out edges mapped to any given core must not exceed 4,096 (N_axout)
- The total number of distribution lists, associated by axon_id, in any core must not exceed 4,096 (N_axin)

The architecture of the neuromorphic cores

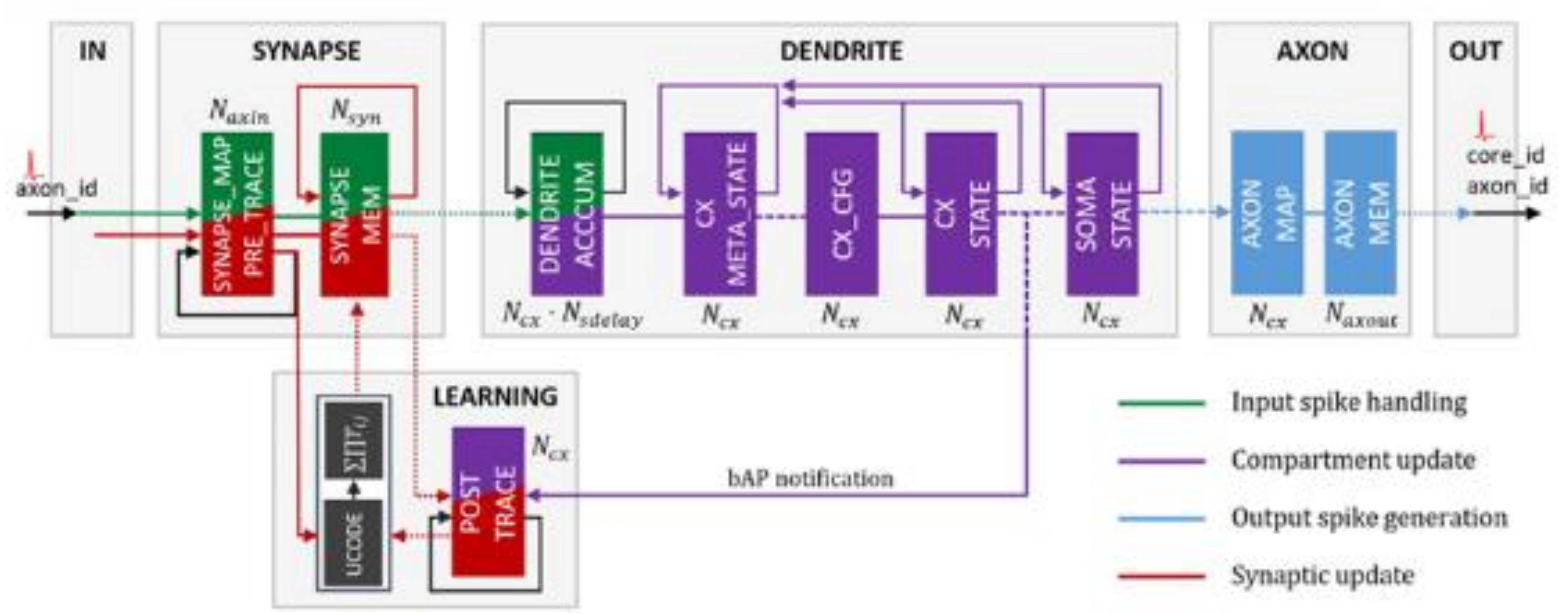


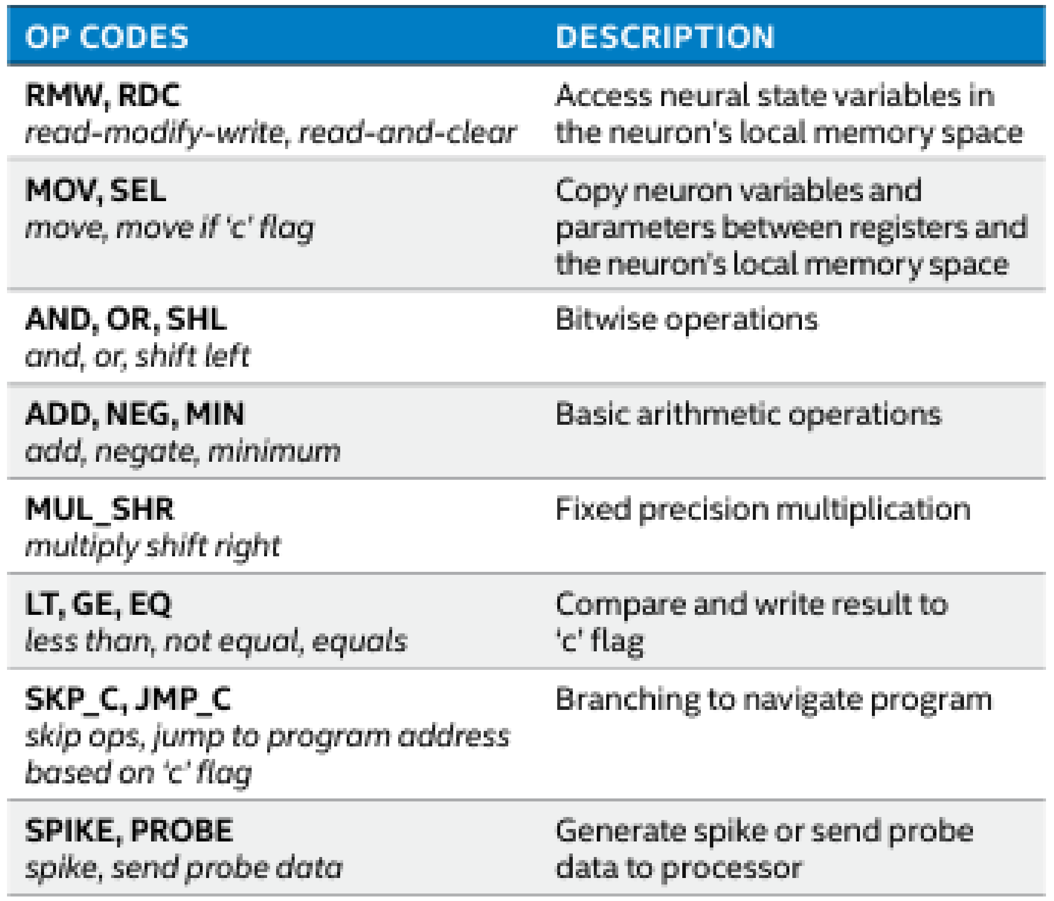
Figure 4. Core Top-Level Microarchitecture. The SYNAPSE unit processes all incoming spikes and reads out the associated synaptic weights from the memory. The DENDRITE unit updates the state variables *u* and *v* of all neurons in the core. The AXON unit generates spike messages for all fan-out cores of each firing neuron. The LEARNING unit updates synaptic weights using the programmed learning rules at epoch boundaries.

Loihi 2. Novelty

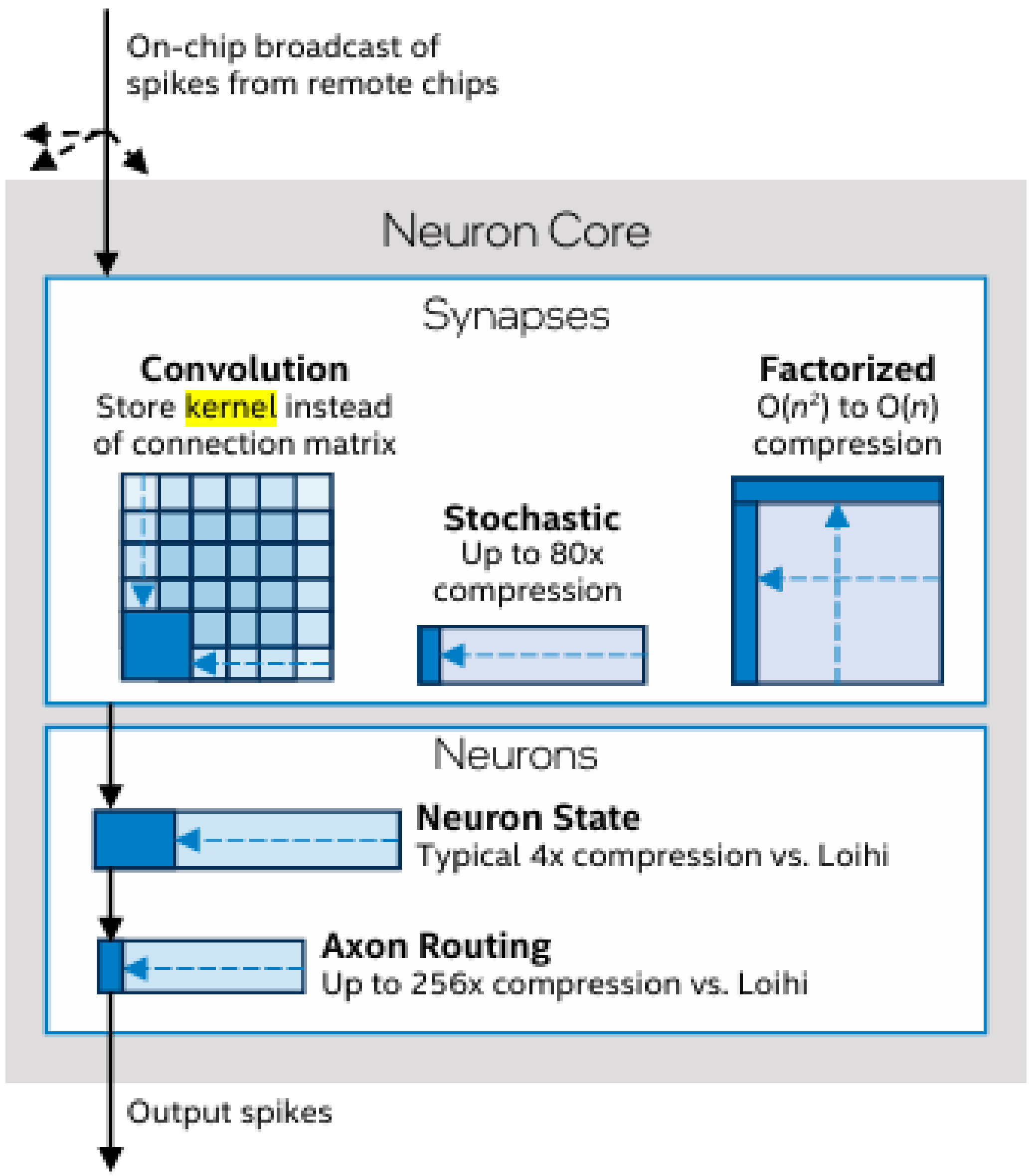
List of the new features:

1. Loihi 2 supports fully programmable neuron models with graded spikes
2. Support for three-factor learning rules

Table 1. Highlights of the Loihi 2 Instruction Set



OP CODES	DESCRIPTION
RMW, RDC *read-modify-write, read-and-clear*	Access neural state variables in the neuron's local memory space
MOV, SEL *move, move if 'c' flag*	Copy neuron variables and parameters between registers and the neuron's local memory space
AND, OR, SHL *and, or, shift left*	Bitwise operations
ADD, NEG, MIN *add, negate, minimum*	Basic arithmetic operations
MUL_SHR *multiply shift right*	Fixed precision multiplication
LT, GE, EQ *less than, not equal, equals*	Compare and write result to 'c' flag
SKP_C, JMP_C *skip ops, jump to program address based on 'c' flag*	Branching to navigate program
SPIKE, PROBE *spike, send probe data*	Generate spike or send probe data to processor


On-chip broadcast of spikes from remote chips
Neuron Core
Synapses
Convolution
Store kernel instead of connection matrix
Stochastic
Up to 80x compression
Factorized
O(n²) to O(n) compression
Neurons
Neuron State
Typical 4x compression vs. Loihi
Axon Routing
Up to 256x compression vs. Loihi
Output spikes

Loihi 2 at a Glance

Table 2 provides a comprehensive comparison of Loihi 2 features versus Loihi features.

Table 2. Comparison of Loihi to Loihi 2

Resources/Features	Loihi	Loihi 2
Process	Intel 14nm	Intel 4
Die Area	60 mm^2	31 mm^2
Core Area	0.41 mm^2	0.21 mm^2
Transistors	2.1 billion	2.3 billion
Max # Neuron Cores/Chip	128	128
Max # Processors/Chip	3	6
Max # Neurons/Chip	128,000	1 million
Max # Synapses/Chip	128 million	120 million
Memory/Neuron Core	208 KB, fixed allocation	192 KB, flexible allocation
Neuron Models	Generalized LIF	Fully programmable
Neuron State Allocation	Fixed at 24 bytes per neuron	Variable from 0 to 4096 per neuron depending on neuron model requirements
Connectivity Features	Basic compression features: • Variety of sparse and dense synaptic compression formats • Weight sharing of source neuron fanout lists	In addition to the Loihi 1 features: • Shared synapses for convolution • Synapses generated from seed • Presynaptic weight-scaling factors • Core fan-out list compression and sharing • Broadcast of spikes at destination chip
Information Coding	Binary spike events	Graded spike events (up to 32-bit payload)
Neuron State Monitoring (for development/debug)	Requires remote pause and query of neuron memory	Neurons can transmit their state on-the-fly
Learning Architecture	Programmable rules applied to pre-, post-, and reward traces	Programmable rules applied to pre-, post-, and generalized "third-factor" traces
Spike Input	Handled by embedded processors	Hardware acceleration for spike encoding and synchronization of Loihi with external data stream
Spike Output	1,000 hardware-accelerated spike receivers per embedded processor	In addition to the Loihi 1 feature, hardware accelerated spike output per chip for reporting graded payload, timing, and source neuron
External Loihi Interfaces	Proprietary asynchronous interface	Support for standard synchronous (SPI) and asynchronous (AER) protocols, GPIO, and 1000BASE-KX, 2500BASE-KX, and 10GBase-KR Ethernet
Multi-Chip Scaling	2D tile-able chip array Single inter-chip asynchronous protocol with fixed pin-count	3D tile-able chip array Range of inter-chip asynchronous protocols with variable pipelining and pin-counts optimized for different system configurations
Timestep Synchronization	Handled by cores	Accelerated by NoC routers

Resonate-and-Fire (RF) model

$$\frac{dz_k}{dt} = (-\gamma + i\omega_k)\, z + a(t)$$

Discrete:

$$z_k[t] \;=\; \lambda e^{i\omega_k \Delta t}\, z[t-1] + a(t)$$

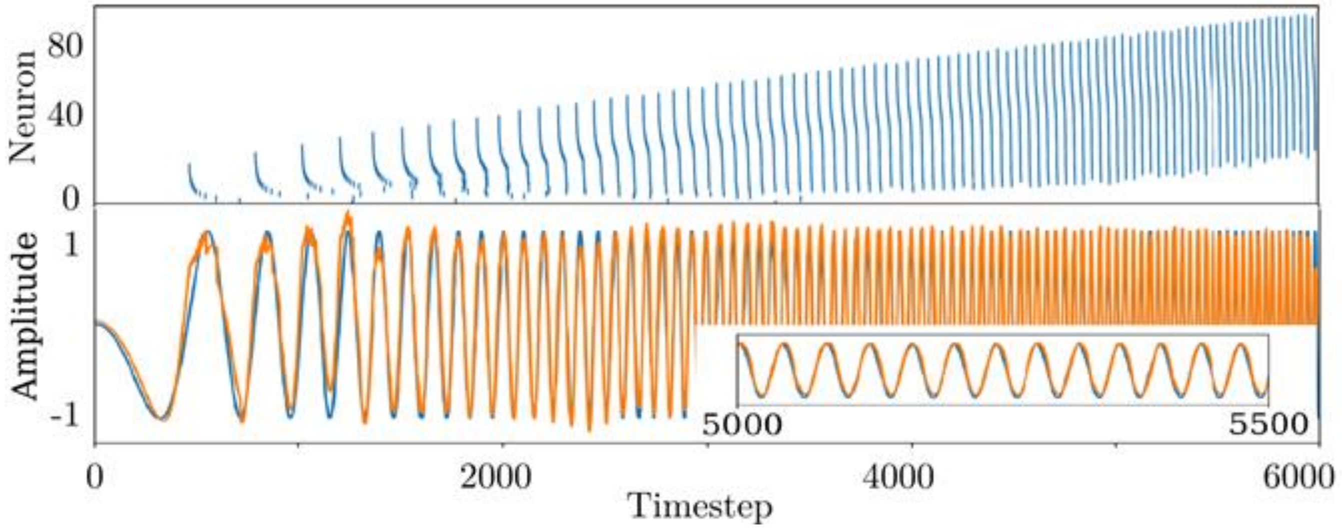
Spiking rule:

$$s[t] = \Re\mathfrak{e}\, z[t] \quad \Longleftrightarrow \quad \Im\mathfrak{m}\, z[t] = 0 \;\text{ and }\; |z| > \theta$$

Given an input a(t), this system approximates its STFT:

$$z_k[0] = 0 \quad \Rightarrow \quad z_k[t] = \sum_{n=0}^{t} \lambda^n e^{in\omega_k \Delta t}\, a(t-n)$$

Figure: spiking patterns of 100-neuron system (top), original and recovered signals (bottom)



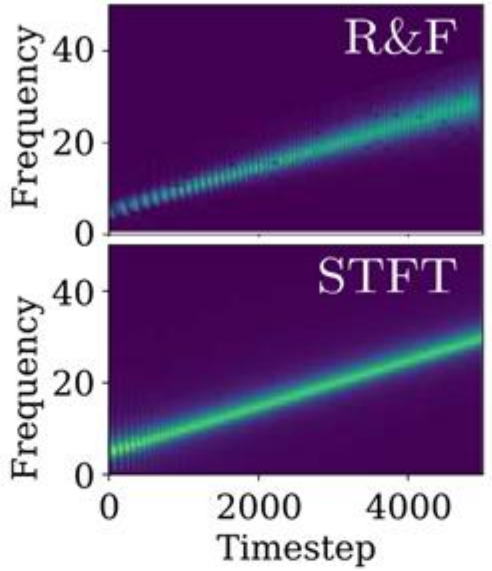


Figure: spectrogram comparison

Estimating Optical Flow with RF

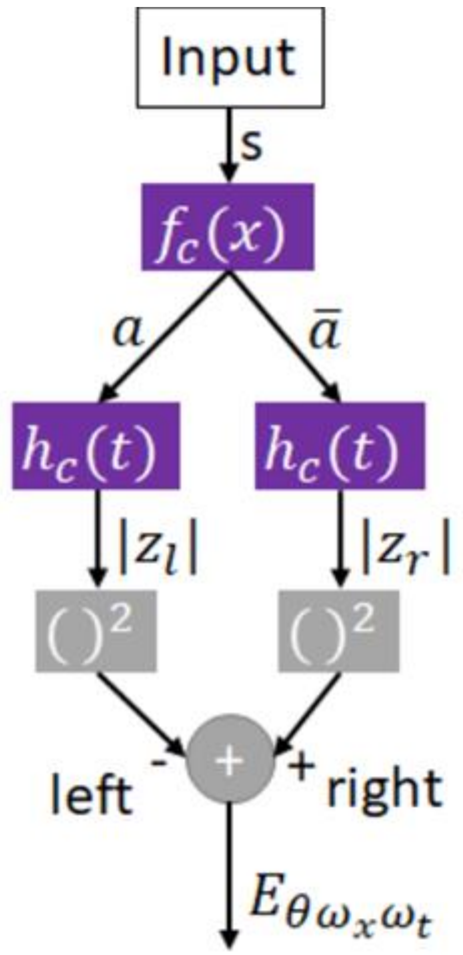
Input – spikes

- sparse event-based DVS data
- In the case of video frames, a separate graded spike is used to represent the intensity of each pixel.

Estimate energy from input.

$f(x)$ – spatial filter

$h(t)$ – temporal filter



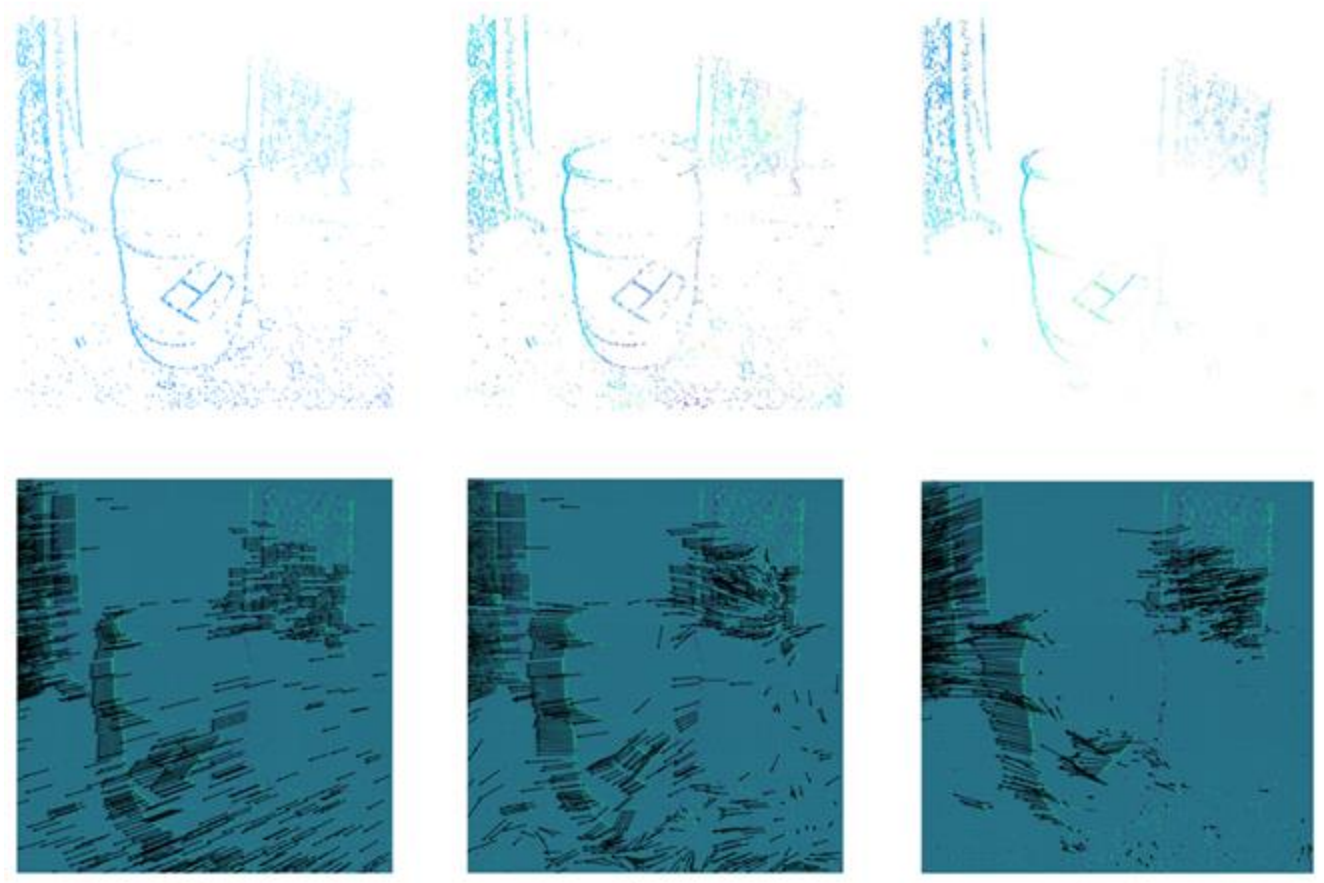
Velocity

$$v_{\omega_x,\omega_t,\theta} = \left[\frac{\omega_t}{\omega_x}\cos\theta, \quad \frac{\omega_t}{\omega_x}\sin\theta\right]$$

Optical flow

$$f = \frac{\sum_{\omega_x,\omega_t,\theta} v_{\omega_x,\omega_t,\theta} E_{\omega_x,\omega_t,\theta}}{\sum_{\omega_x,\omega_t,\theta} E_{\omega_x,\omega_t,\theta}}$$

Estimating Optical Flow with RF. Results



Ground Truth Flow | Ours | EV-FlowNet

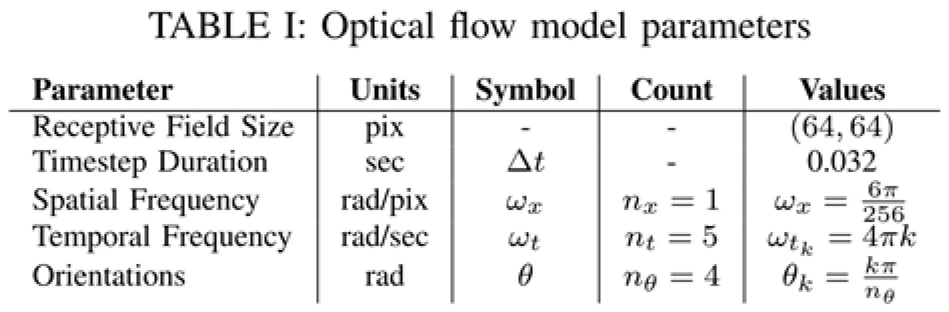


TABLE I: Optical flow model parameters

Parameter	Units	Symbol	Count	Values
Receptive Field Size	pix	-	-	$(64, 64)$
Timestep Duration	sec	Δt	-	0.032
Spatial Frequency	rad/pix	ω_x	$n_x = 1$	$\omega_x = \frac{6\pi}{256}$
Temporal Frequency	rad/sec	ω_t	$n_t = 5$	$\omega_{t_k} = 4\pi k$
Orientations	rad	θ	$n_\theta = 4$	$\theta_k = \frac{k\pi}{n_\theta}$

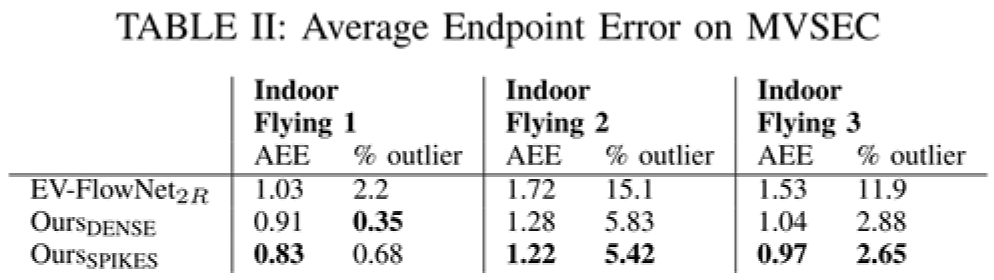


TABLE II: Average Endpoint Error on MVSEC

	Indoor Flying 1		**Indoor Flying 2**		**Indoor Flying 3**	
	AEE	% outlier	AEE	% outlier	AEE	% outlier
EV-FlowNet$_{2R}$	1.03	2.2	1.72	15.1	1.53	11.9
Ours$_{\text{DENSE}}$	0.91	**0.35**	1.28	5.83	1.04	2.88
Ours$_{\text{SPIKES}}$	**0.83**	0.68	**1.22**	**5.42**	**0.97**	**2.65**

SLAYER

$$E = \tfrac{1}{2}\int_0^T \left(e^{(n_l)}(t)\right)^2 dt, \quad e^{(n_l)}(t) = \left(\varepsilon * \left(s^{(n_l)} - \hat{s}\right)\right)(t)$$

$$s_i^{(l)}(t) = \sum_f \delta\!\left(t - t_{i,f}^{(l)}\right)$$

$$e^{(l)}(t) = \begin{cases} \dfrac{\partial L(t)}{\partial a^{(n_l)}(t)} & l = n_l \\ \left(W^{(l)}\right)^{\top} \delta^{(l+1)}(t) & \text{otherwis} \end{cases}$$

$$a_i^{(l)}(t) = \left(\varepsilon_d * s_i^{(l)}\right)(t)$$

$$u_j^{(l+1)}(t) = \sum_i W_{ji}^{(l)}\, a_i^{(l)}(t) + \left(\nu * s_j^{(l+1)}\right)(t)$$

$$\rho^{(l)}(t) = \alpha^{-1} \exp\!\left(-\beta \left|u^{(l)}(t) - \vartheta\right|\right)$$

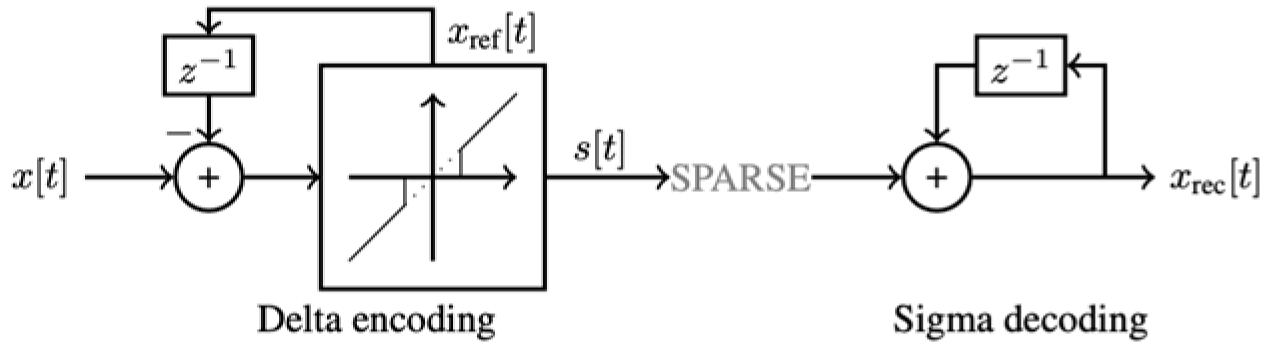
$$\delta^{(l)}(t) = \rho^{(l)}(t)\left(\varepsilon_d \odot e^{(l)}\right)(t)$$

$$s_j^{(l+1)}(t) = f_s\!\left(u_j^{(l+1)}(t)\right)$$

$$\nabla_{W^{(l)}} E = \int_0^T \delta^{(l+1)}(t)\left[a^{(l)}(t)\right]^{\top} dt \quad \nabla_{d^{(l)}} E = -\int_0^T \dot{a}^{(l)}(t)\, e^{(l)}(t)\, dt$$

Efficient Video and Audio Processing with LOIHI 2

Since Intel Loihi 2 is suitable for RF networks and **Sigma-Delta encapsulation**:



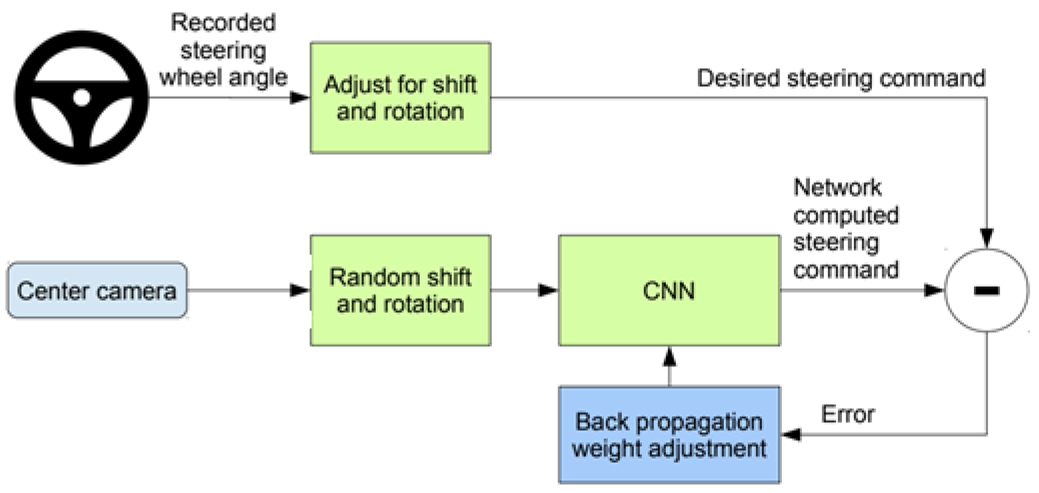
$$s[t] = (x[t] - x_{\text{ref}}[t-1])\,\mathcal{H}(|x[t] - x_{\text{ref}}[t-1]| - \vartheta) \qquad (1)$$
$$x_{\text{ref}}[t] = x_{\text{ref}}[t-1] + s[t].$$

$$x_{\text{rec}} = x_{\text{rec}}[t-1] + s[t] \qquad (2)$$

Authors integrate these concepts into regular ANNs and **benchmark** Loihi 2 SNNs on Video and Audio tasks in terms of **Efficiency** and **Delay** **VS** NVidia Jetson and CPU

Efficient Video and Audio Processing with LOIHI 2

Video task setup:



Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Network computed steering command
Center camera
Random shift and rotation
CNN
Back propagation weight adjustment
Error

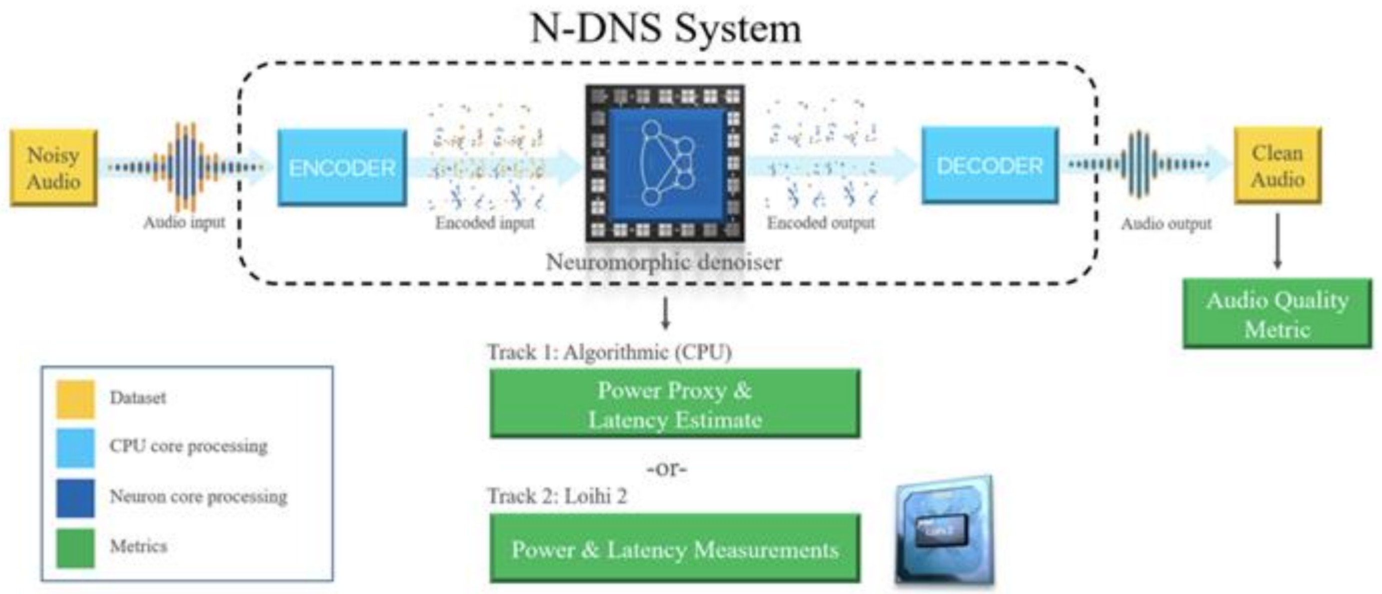
Figure 2: Training the neural network.

The consecutive frames have high degree of temporal correlation -> Loihi 2 utilizes Sigma-Delta ReLU neurons.

MSE angle results(↓): <u>0.035</u> for Loihi 2 vs <u>0.025</u> 8-bit quant. ANN (NVidia Jetson Nano)

Efficient Video and Audio Processing with LOIHI 2

Audio denoising task setup:



Metric:

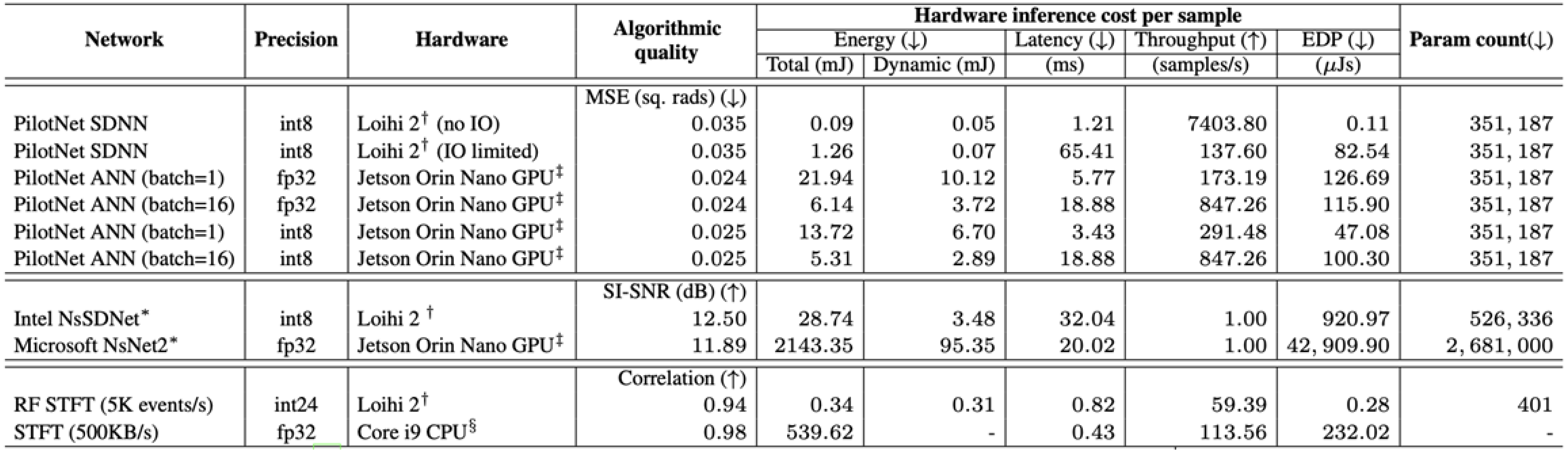
$$\text{SI-SNR} := 10 \log_{10} \frac{||s_{\text{target}}||^2}{||e_{\text{noise}}||^2},$$

$$\text{where } s_{\text{target}} := \frac{\langle \hat{s}, s \rangle s}{||s||^2} \text{ and } e_{\text{noise}} := \hat{s} - s_{\text{target}}$$

Audio data is temporal and dense, 16kHz sampling rate means 16k measurements per second!

SI-SNR results(↑) : <u>12.5</u> for Loihi 2 vs <u>11.89</u> ANN baseline (NVidia Jetson Nano)

Efficient Video and Audio Processing with LOIHI 2

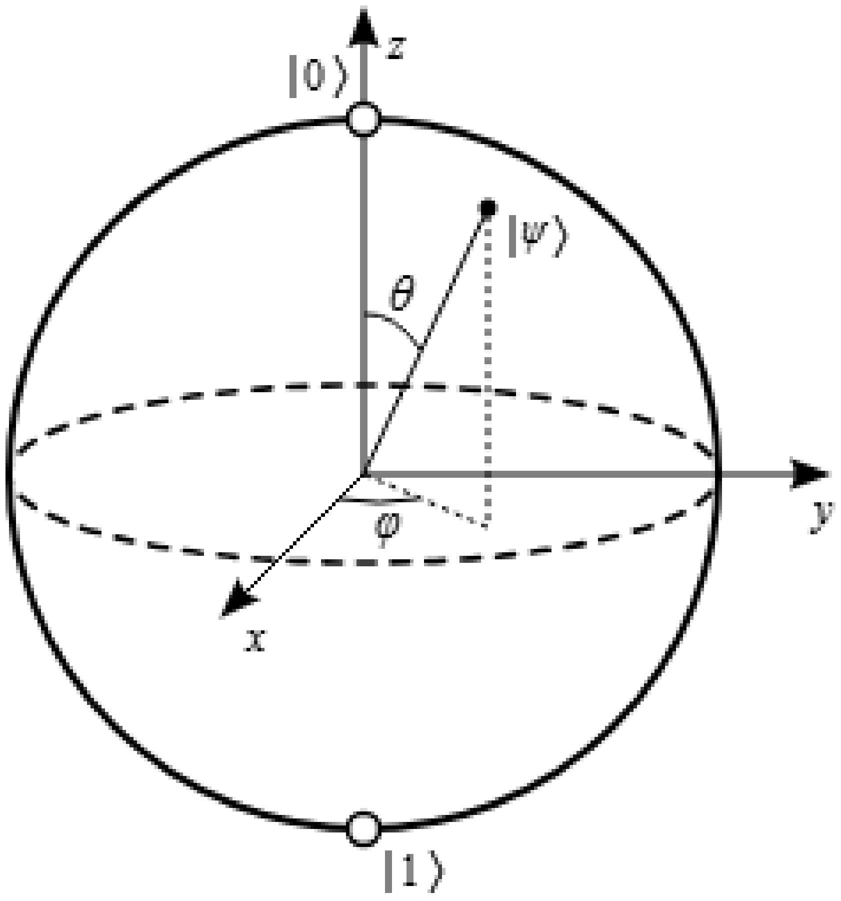


Network	Precision	Hardware	Algorithmic quality	Hardware inference cost per sample					Param count(↓)
				Energy (↓)		Latency (↓)	Throughput (↑)	EDP (↓)	
				Total (mJ)	Dynamic (mJ)	(ms)	(samples/s)	(μJs)	
			MSE (sq. rads) (↓)						
PilotNet SDNN	int8	Loihi 2[†] (no IO)	0.035	0.09	0.05	1.21	7403.80	0.11	351, 187
PilotNet SDNN	int8	Loihi 2[†] (IO limited)	0.035	1.26	0.07	65.41	137.60	82.54	351, 187
PilotNet ANN (batch=1)	fp32	Jetson Orin Nano GPU[‡]	0.024	21.94	10.12	5.77	173.19	126.69	351, 187
PilotNet ANN (batch=16)	fp32	Jetson Orin Nano GPU[‡]	0.024	6.14	3.72	18.88	847.26	115.90	351, 187
PilotNet ANN (batch=1)	int8	Jetson Orin Nano GPU[‡]	0.025	13.72	6.70	3.43	291.48	47.08	351, 187
PilotNet ANN (batch=16)	int8	Jetson Orin Nano GPU[‡]	0.025	5.31	2.89	18.88	847.26	100.30	351, 187
			SI-SNR (dB) (↑)						
Intel NsSDNet[*]	int8	Loihi 2 [†]	12.50	28.74	3.48	32.04	1.00	920.97	526, 336
Microsoft NsNet2[*]	fp32	Jetson Orin Nano GPU[‡]	11.89	2143.35	95.35	20.02	1.00	42, 909.90	2, 681, 000
			Correlation (↑)						
RF STFT (5K events/s)	int24	Loihi 2[†]	0.94	0.34	0.31	0.82	59.39	0.28	401
STFT (500KB/s)	fp32	Core i9 CPU[§]	0.98	539.62	-	0.43	113.56	232.02	-

Results for Intel Loihi 2:

- PilotNet Steering: up to 150x less Energy, 2.8x faster latency
- Noise suppression: up to 74x less Energy, better SI-SNR, 526k vs 2.68M params
- STFT: up to 1500x less Energy, 828x better EDP

Briefly about Quantum Computing



Hadamard	$-\boxed{H}-$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
Pauli-X	$-\boxed{X}-$	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$
Pauli-Y	$-\boxed{Y}-$	$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$
Pauli-Z	$-\boxed{Z}-$	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$