SpiNNaker

Limitations au niveau hardware

- 16 processeurs de calcul par puce.
- Mémoire /cœur limitée (64 Ko de données et 32 Ko d'instruction)
- Pas de floating point (« The SpiNNaker platform has no hardware floating-point support, thus the neuron states and synapses are computed using fixed-point arithmetic (Furber et al., 2013, 2014). This was part of the design specification in order to further improve the energy efficiency of the platform. » Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms)
- 128 Mo de SDRAM. Les données nécessaires à l'exécution sont stockées dans SDRAM. Cela laisse environ 8Mo par cœur pour stocker les paramètres de la population de neurones et les synapses entre les neurones.

Limitations au niveau software

Utilisation de PyNN : limité car SpiNNaker ne supporte pas tout ce qui est défini dans PyNN. Pour n'utiliser que les fonctions et modèles supportés par la machine SpiNNaker : utilisation de spyNNaker, qui implémente la partie de PyNN v0.7 fonctionnelle.

Version de PyNN : spyNNaker implémente une partie de PyNN 0.7 (et pas la dernière version v0.8).

Modèles : sPyNNaker supporte seulement les modèles :

Leaky intergrate and fire current exponential IFCurrExp
Leaky intergrate and fire conductive exponential IFCondExp
Leaky intergrate and fire duel current exponential IFCurrDualExp
Izhikevich Current Exponential Population IZKCurrExp
External Input

Pour injecter des spikes : SpikeSourceArray ou SpikeSourcePoisson

Connecteurs : AllToAllConnector
DistanceDependentProbabilityConnector
FixedNumberPreConnector
FromFileConnector
OneToOneConnector
MultapseConnector
FixedProbabilityConnector
FromListConnector

Plasticity : sPyNNaker supporte actuellement la plasticité décrite par STDPMechanism which is set as the slow property of SynapseDynamics.

sPyNNaker supporte les STDP timing dependence rules: PfisterSpikeTripletRule and SpikePairRule and les STDP weight dependence rules: AdditiveWeightDependence and MultiplicativeWeightDependence

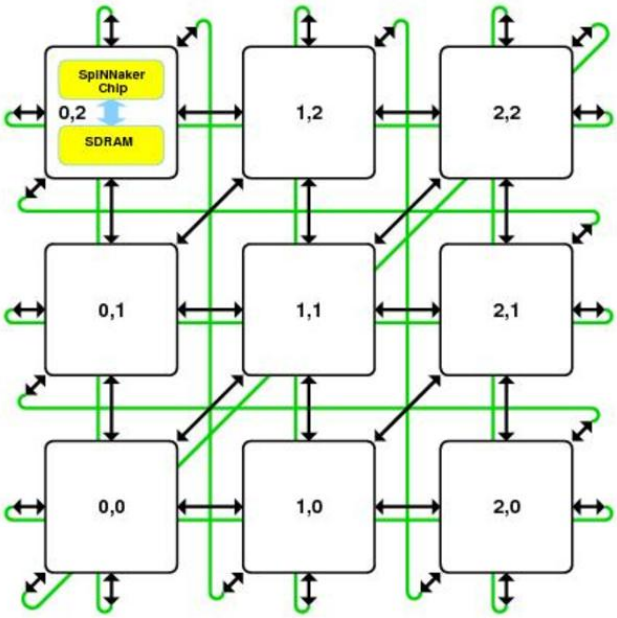
1000 neurones/cœur possibles sur le matériel mais limitation à 256 neurones/cœur et 256 synapses/neurone due à la façon dont les delay sont implémentés dans spyNNaker. Tous les modèles supportent des delay entre 1 timestep et 144 timesteps. Si delay > 16 time steps (16ms si 1 time step = 1ms), alors chaque atome aura un modèle de delay associé, ce qui prend un cœur sur la machine par delay extension, ce qui réduit le nombre de neurones simulables.

« All of our neural models have a limitation of 256 neurons per core. »
« All our models support delays between 1 timestep and 144 timesteps. Delays of more than 16 timesteps are supported by delay extensions which take up another core within the machine, thus use of such delays will further limit the total number of neurons that can be supported in any simulation. »
https://spinnakermanchester.github.io/2015.005.Arbitrary/SPyNNakerLimitations.html

Les populations de plus de 256 neurones sont automatiquement fragmentées.

Remarques :

- spyNNaker propose un front end en mode virtuel afin de tester la simulation avant de l'exécuter en vrai.
- Il est possible de spécifier le nombre de neurones par coeur : p.set_number_of_neurons_per_core(p.IF_curr_exp, 100) et de préciser le cœur qui gérera la population : pop.add_placement_constraint(x=1, y=1).
- Chaque puce a une adresse fixe et chaque cœur est repérable avec un x,y. Chaque puce communique avec les autres par 6 liens bidirectionnels composé de 7 data wire+ 1 acknowledge wire :



Énergie

http://www.artificialbrains.com/spinnaker
« The SpiNNaker machine is expected to consume 50-100 kW peak, although the average is predicted to be well below 50 kW. For comparison, the average human brain consumes around 20 W. »

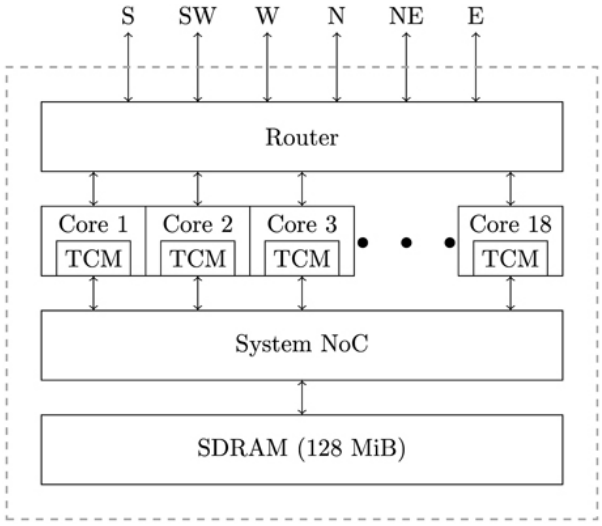
Power_analysis_of_large_scale_real_time_neural_networks_on_SpiNNaker.pdf
« We present as contributions, the simulation of large-scale real-time simulations of up to a quarter of a million neurons generating more than billion synaptic events per second, with each SpiNNaker chip in the simulation consuming less than 1 W. »

(http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6515159)
« We estimate that a full-size million-core SpiNNaker will consume around 90 kW, which is just within the range of off-the-shelf forced-air cooling systems. »

Power consumption : 1W/puce si pic, 360mW/puce si repos
20 mW/cœur au repos
SDRAM : 170 mW

Fonctionnement par couches
3 couches : matériel (puces, cœurs) / applicatif bas niveau / applicatif haut niveau

Matériel : cf présentation SpiNNaker du 07/11
Au centre de chaque puce, il y a routeur multicast qui communique les spikes intra-chip et inter-chip. Chaque cœur a un communication controller qui génère et reçoit les paquets du routeur. Plusieurs routages sont possibles : multicast, point à point , au plus proche, route fixe, … Il existe un routage d'urgence : redirection temporaire sur les voisins si fail/congestion



Software bas niveau: (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4458614/)
2 parties : sur la puce et sur l'hôte.

- Sur l'hôte (workstation qui contrôle le système) : I/O et monitoring. Ybug propose une interface de commande et de debug.

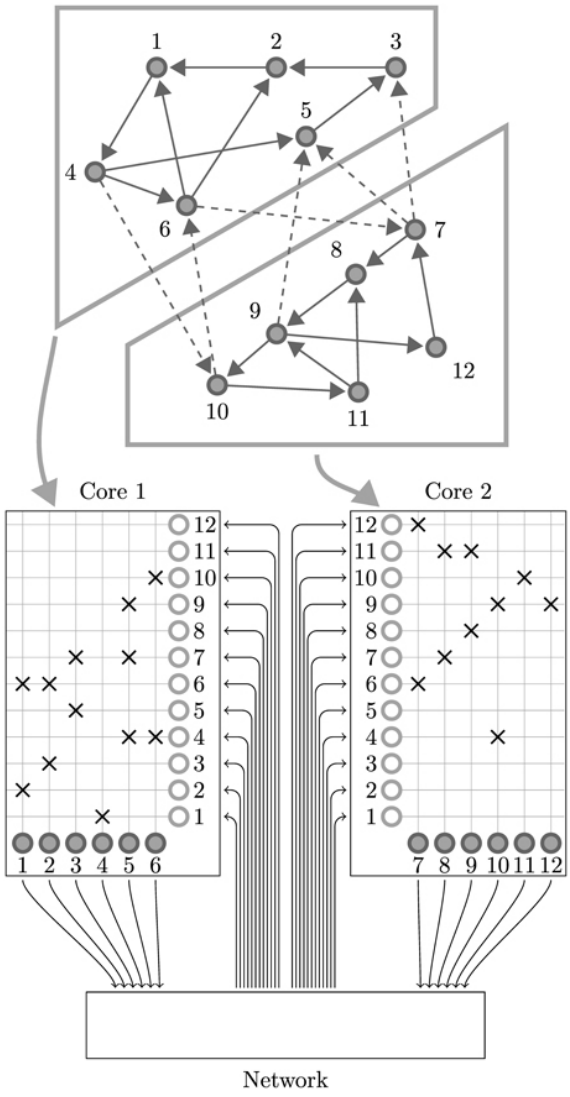
- Sur la puce:

  Coeur moniteur : Scamp (SpiNNaker Control and Monitor Program) interagit avec ybug et sark, communication entre les processeurs et communication avec l'hôte, exécute une application qui collecte les spikes et les envoie à l'hôte, chargé de collecter les résultats de la simulation. Communication de paquets via protocole SDP.

  Cœurs applicatifs : exécutent SARK (SpiNNaker Application Run-time Kernel), qui exécute l'application et distribue/schedule les callbacks des fonctions, avec 2 threads : 1 scheduler et un dispatcher. Si pas de tâches à exécuter, passe en mode low power sleep.

Software haut niveau :
PyNN (spyNNaker) permet de décrire des topologies neuronales avec des populations de neurones et des projections.

PACMAN (Partition and configuration management) : mappe la description des neurones PyNN sur les cœurs de la machine, selon les ressources disponibles et gère le routage. La répartition est automatique mais l'utilisateur peut spécifier des contraintes dans le script PyNN. PACMAN possède plusieurs algorithmes de routage et il est possible de définir ses propres algorithmes.
1) assigner chaque neurone à un processeur
2) map chaque neurone dans une adresse virtuelle
3) génération des infos de routage: une route par connexion définie, route doit passer par le moins de routeurs possible et il doit y avoir le moins de routes possibles.



SpiNNmachine : représentation de la machine SpiNNaker. Représentation de l'état actuel / aperçu de la machine sur laquelle la simulation va être exécutée.

SpiNNMan : communiquer avec SpiNNaker : requête sur l'état de la machine (cœur/puces dispos, cœur running ou non, boot correct, ….)

Ethernet pour : load software/data, output informations, …

PyNN

Populations	groupe de neurones du même type
View	vue d'un sous ensemble d'une population. Non supporté par spynnaker
Assembly	groupe de neurones, peut être hétérogène. Non supporté par spynnaker
Connecteurs	synapses
Projection	contient toute les connexions d'un certain type entre 2 populations

StandardCellType : classe de base dont hérite tous les modèles de neurones
Spike sources : génère des spikes selon différents paramètres.
Connexion synaptique entre deux neurones : classe « synapse type ». Modélise le retard, le poids, le comportement (plasticité). Classe standard, StandardSynapseType, staticSynapse, shorttermplasticity, …

Simulation control : setup, run (avance la simulation de x ms), run_until (ms), reset, end, get/set, record
RandomDistribution : spécifier une distribution, NumpyRNG = Mersenne Twister ou NativeRNG : dépend du simulateur (spiNNaker : Mersenne twister)
SpatialStructure : représentation de la distribution des neurones en 1D, 2D, 3D, randomly
Utility : get_simulator, timer, notify (envoyer un email quand la simulation est finie), save_population, load_population, ...

Stokes_The_SpiNNaker_Software_Stack_2.pdf:
Exemple de synfire chain : activité qui se propage à travers le réseau

1) Importer les bibliothèques (spiNNaker as sim, nimpy.random, …)

import pyNN.spiNNaker as p

2) Définir la population, les propriétés des neurones, des synapses, … Choisir le type de cellule (Integrate and fire), la population (collection de neurones tous du même type), les connexions (synapses, par exemple : OneToOne : connexion des neurones de deux populations deux à deux)

pop = p.Population(1, p.IF_curr_exp, {}, label="pop") *// population de neurones*
input = p.Population(1, p.SpikeSourceArray, {'spikes_times':[0]}, label="input") *// Source spikes*
input_proj = p.Projection(input, pop, p.OneToOneConnector(weights=5.0, delays=1)) *//synapses*

3) Setup, enregistrement des spikes dans un fichier

p.setup(timestep=1.0) *// pas de temps de la simulation*
pop.record()
pop.record_v()

4) Run. On peut ensuite utiliser des outils pour analyser et visualiser les résultats.

p.run(10)