(Dossier Events)

A GALS Infrastructure for Massively Parallel Multiprocessor

1 nœud = 1 puce multiprocesseurs + une puce mémoire
= 20 cœurs ARM9 (choisis pour leur "high power efficiency", à 200MHz), une interface mémoire, un routeur multicast (route les spikes entre les cœurs) et 2 Network on Chip pour la communication entre les cœurs internes à la puce et l'environnement.

6 liens bidirectionnels relient les puces entre elles (cf schéma) --> permet plusieurs topologies. Utilise des canaux autonomes, coûteux en terme de fil mais nettement plus efficaces en énergie que des liaisons synchrones de bande passante similaire. 6 liens: topologie torus

NoC asynchrones --> nœuds asynchrones
Cœurs et RAM synchrones à l'intérieur de la puce
GALS approche --> découple problèmes d'horloge --> plus efficace niveau énergie

SpiNNaker : but : simuler un milliard de neurones en temps réel
Pas nécessaire de garder cohérence dans mémoire partagée
Communication se fait par "spikes" : quand neurone simulé au delà d'un certain seuil, le spike est communiqué à tous les neurones voisins (environ 1000). En théorie, chaque processeur peut modéliser 1000 neurones.

A l'initialisation, les processeurs font un "self test". Le premier à avoir réussi devient le "Moniteur" et s'occupe de la gestion des autres cœurs.

Chaque cœur a 100Ko de mémoire locale sur la puce. Mémoire partagée SDRAM (=la puce mémoire) de 128 Mo utilisée surtout pour stocker les "neural weights".

Possible de modéliser réseaux en 2, 3 ou + dimensions. SpiNNaker mappe chaque neurone dans une adresse virtuelle.. Allocations peuvent être arbitraires bien que les affectations tenant compte de la structure physique peuvent améliorer l'efficacité.

Les neurones peuvent être alloués à n'importe quel processeur, et les tables de routage doivent être configurées pour envoyer les événements neuronaux en conséquence.

Bus pour connecter tout sur une puce de 100mm² --> challenge car bus synchrones conventionnels --> problème de bottleneck si 20 cœurs accèdent au bus.
Approche globalement asynchrone, localement synchrone (GALS) permet à chaque bloc synchrone de s'exécuter dans son propre domaine temporel. La puce utilise 2 Network on Chip (NoCs).

Le système NoC remplace un bus conventionnel pour l'interconnexion du système. Connecte 20 cœurs + routeur à autre matériel dont la SDRAM. La bande passante pour l'accès à la mémoire SDRAM est de 1 Go/s. Chaque cœur a un contrôleur d'accès à la mémoire dédié à déplacer des données de et à partir de la SDRAM. Plus couteux qu'un bus mais permet à tout cœur de communiquer avec la SDRAM alors qu'un autre cœur communique avec n'importe lequel des autres composants du système.

Le NoC communication fournit une infrastructure de commutation de paquets sur et hors-puce. Transporte les spikes, les infos de configuration et de monitoring.

Routeur : route tous les paquets qui arrivent sur ses inputs. Routing table multicats ou point à point ou nearest-neighbor or emergency, ...
Identifie et traite les erreurs

Énergie:
Avec un système de cette envergure, la consommation énergétique peut rapidement poser problème. De 0,12 mW/MHz à 0,23 mW/MHz pour un noyau ARM968 dans une technologie de process de 130 nm. Chaque puce Spinnaker consommera de 250 mW à 500 mW. La puissance requise pour la communication est négligeable, chaque paquet consommant 1 nJ pour chaque routeur et 1 nJ pour chaque liaison interchip qu'il traverse. **Un système à grande échelle capable de simuler un milliard de neurones en temps réel nécessitera 50 000 nœuds et consommera de 23 kW à 36 kW.**

Managing Burstiness and Scalability in Event-Driven Models on the SpiNNaker Neuromimetic System

Parallélisme natif : plusieurs processeurs par puce, chacun complètement indépendant des autres (différentes horloges). Chaque nœud utilise uniquement les informations locales pour controler l'exécution et opère de façon asynchrone par rapport aux autres nœuds.

Event-Driven Processing: An external, self-contained, instantaneous signal drives state change in each process. Spikes fonctionnent comme des événements, envoyés par un neurone en spécifiant l'expéditeur mais pas le destinataire.

Incohérence mémoire : chaque processeur peut modifier la mémoire qui lui est accessible sans notifier ou se synchroniser avec les autres processeurs. Comme en biologie, utilise seulement information locale. Processeurs ont accès à 2 mémoire : leur propre mémoire locale (fortement couplée) et la mémoire SDRAM de la puce. Pas de mécanisme de cohérence et/ou de synchronisation. La mémoire locale au processeur n'est accessible que par lui. Dans la SDRAM : synaptic data. Comme chaque synapse dans la SDRAM se connecte à un seul neurone cible résidant dans un processeur spécifique, la SDRAM est segmentée en régions discrètes pour chaque processeur, regroupées par neurones postsynaptiques. Ceci évite la nécessité d'une vérification de cohérence, car un seul nœud de processeur accède à une plage d'adresses donnée. Ainsi, SDRAM se comporte plus comme une extension à la mémoire locale que comme une mémoire partagée. Du point de vue du système, toute la mémoire est locale.

Incremental Reconfiguration: The structural configuration of the hardware can change dynamically while the system is running (plastic). Routeur multicast : reproduit neural fan out : environ 1000 connexions par neurone. Le routage est entièrement reprogrammable en modifiant la table de routage, le modèle simulé est entièrement reprogrammable en changeant le code à exécuter. Il est possible, au moins en principe, de reconfigurer le modèle durant l'exécution.

Non deterministic process dynamics :

Communication asynchrones --> état global du système ne signifie rien, impossible d'avoir un instantané du système

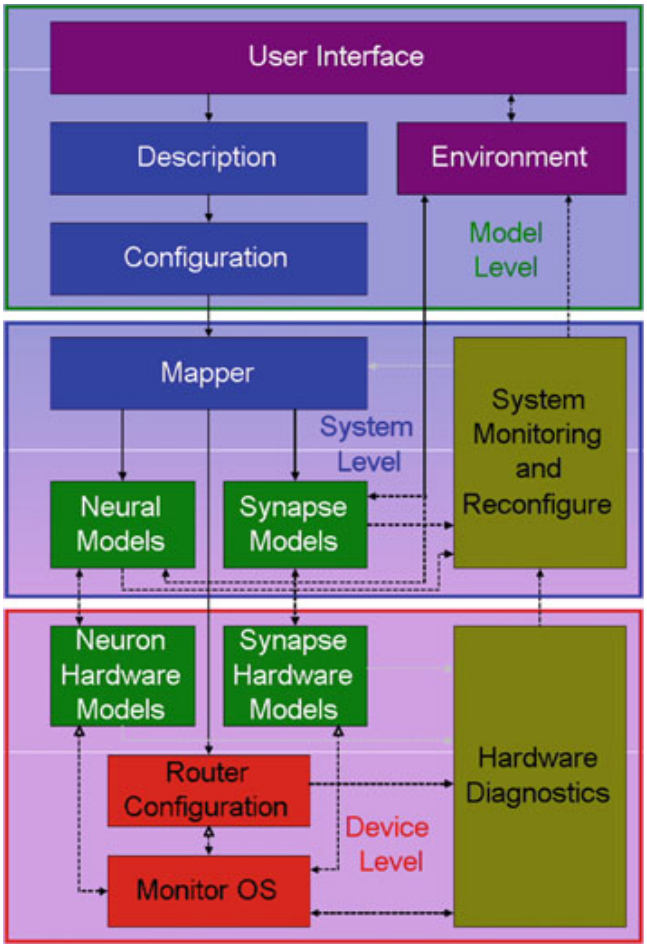
One way communication : source routed, "fire and forget", use it or lose it

Pas de contrôle des ressources partagées : chaque processeur a accès aux ressources

partagées et est indépendant des autres --> Memory model incoherent

Ainsi, trois axiomes importants pour le parallélisme ne sont pas respectés ici sans compromettre la capacité de réaliser des calculs significatifs : le déterminisme, la cohérence mémoire et la synchronicité.

3-level software model : model, system, device. Model : abstraction, masque le hardware.



Pour des réseaux sans spike : time sampling : les événements ne peuvent arriver qu'à des intervalles de temps fixes. Pour les modèles sans composante temps : quand un neurone reçoit un input, il utilise l'input pour faire son calcul et envoie le output comme un événement à son neurone cible.

Code : synfire_chain

Import : spiNNaker comme simulateur, numpy.random pour nombres aléatoires, pyplot pour dessiner graphique des résultats.

```
import pyNN.spiNNaker as sim
import numpy.random
import matplotlib.pyplot as plt
```

11 populations de 8 neurones

```
n_populations = 11
population_size = 8
```

```
neuron_parameters = { 'cm': 0.2,  'v_reset': -70, 'v_rest': -70,  'v_thresh': -47, 'e_rev_I': -70,
'e_rev_E': 0.0}

weight_exc_exc = 0.005
weight_exc_inh = 0.005
weight_inh_exc = 0.5
delay = 3.0
rng_seed = 42
stimulus_onset = 25.0
stimulus_sigma = 0.5
runtime = 150.0

sim.setup(timestep=0.1)

populations = {'exc': [], 'inh': []}
for syn_type in ('exc', 'inh'):
   populations[syn_type] = [sim.Population(population_size, sim.IF_cond_exp, neuron_parameters)
                            for i in range(n_populations)]

connector_exc_exc = sim.AllToAllConnector(weights=weight_exc_exc, delays=delay)
connector_exc_inh = sim.AllToAllConnector(weights=weight_exc_inh, delays=delay)
connector_inh_exc = sim.AllToAllConnector(weights=weight_inh_exc, delays=delay)

for i in range(n_populations):
   j = (i + 1) % n_populations
   prj_exc_exc = sim.Projection(populations['exc'][i], populations['exc'][j],
                                connector_exc_exc, target='excitatory')
   prj_exc_inh = sim.Projection(populations['exc'][i], populations['inh'][j],
                                connector_exc_inh, target='excitatory')
   prj_inh_exc = sim.Projection(populations['inh'][i], populations['exc'][i],
                                connector_exc_exc, target='inhibitory')

numpy.random.seed(rng_seed)
stim_spikes = numpy.random.normal(loc=stimulus_onset,
                                  scale=stimulus_sigma,
                                  size=population_size)
stim_spikes.sort()
stimulus = sim.Population(1, sim.SpikeSourceArray, {'spike_times': stim_spikes})

prj_stim_exc = sim.Projection(stimulus, populations['exc'][0],
                              connector_exc_exc, target='excitatory')
prj_stim_inh = sim.Projection(stimulus, populations['inh'][0],
                              connector_exc_inh, target='excitatory')
```

Enregistrement

```
for syn_type in ('exc', 'inh'):
   for population in populations[syn_type]:
      population.record()
```

```
sim.run(runtime)

colours = {'exc': 'r', 'inh': 'b'}

id_offset = 0
for syn_type in ['exc', 'inh']:
    for population in populations[syn_type]:
        spikes = population.getSpikes()
        colour = colours[syn_type]
        plt.plot(spikes[:,1], spikes[:,0] + id_offset, ls='', marker='o', ms=1, c=colour, mec=colour)
        id_offset += population.size

plt.xlim((0, runtime))
plt.ylim((-0.5, 2* n_populations * population_size + 0.5))
plt.xlabel('time (t)')
plt.ylabel('neuron index')
plt.savefig("synfire_chain.png")
```

Fichier de configuration pour l'exécution virtuelle :

.spynnaker.cfg

```
[Machine]
width=100
height=100
virtual_board=true
```

Fichiers générés :

Table de routage :

```
Router contains 4 entries
Index Key       Mask       Route      Default [Cores][Links]
----- ---------- ---------- ---------- ------- --------------
    0 0x00000004 0xFFFFFFFF 0x00000280 False   [1, 3] []
    1 0x00000002 0xFFFFFFFE 0x00000080 False   [1] []
    2 0x00000001 0xFFFFFFFF 0x00000180 False   [1, 2] []
    3 0x00000000 0xFFFFFFFF 0x00000080 False   [1] []
0 Defaultable entries
```

Table de routage compressée :

```
Router contains 3 entries
Index Key       Mask       Route      Default [Cores][Links]
----- ---------- ---------- ---------- ------- --------------
 0 0x00000004 0xFFFFFFFF 0x00000280 False   [1, 3] []
 1 0x00000001 0xFFFFFFFF 0x00000180 False   [1, 2] []
 2 0x00000000 0xFFFFFFFC 0x00000080 False   [1] []
0 Defaultable entries
```

Utilisation de la mémoire/cœur :

Memory Usage by Core
==========
Generated: Thu Dec 8 13:10:30 2016 for target machine 'virtual'

SDRAM requirements for core (0,0,1) is 1024 KB
SDRAM requirements for core (0,0,4) is 1024 KB
SDRAM requirements for core (0,0,3) is 0 KB
SDRAM requirements for core (0,0,5) is 1024 KB
SDRAM requirements for core (0,0,2) is 0 KB
**** Chip: (0, 0) has total memory usage of 3072 KB out of a max of 117 MB

Placement Information by Vertex
==========

Generated: Thu Dec 8 13:10:30 2016 for target machine 'virtual'

**** Vertex: 'Population 0'
Model: IFCondExp
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms)

**** Vertex: 'Population 1'
Model: SpikeSourceArray
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms)

**** Vertex: 'Population 1_delayed'
Model: DelayExtensionVertex
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms)

**** Vertex: 'Population 2'
Model: SpikeSourceArray
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms)

**** Vertex: 'Population 2_delayed'
Model: DelayExtensionVertex
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms)

Network Specification

```
====================

Generated: Thu Dec  8 13:10:30 2016 for target machine 'virtual'

*** Vertices:
Vertex Population 0, size: 1
Model: IFCondExp
constraint: PartitionerMaximumSizeConstraint(size=255)

Vertex Population 1, size: 1
Model: SpikeSourceArray
constraint: PartitionerMaximumSizeConstraint(size=2147483647)
constraint: PartitionerSameSizeAsVertexConstraint(vertex=Population 1_delayed)

Vertex Population 2, size: 1
Model: SpikeSourceArray
constraint: PartitionerMaximumSizeConstraint(size=2147483647)
constraint: PartitionerSameSizeAsVertexConstraint(vertex=Population 2_delayed)

Vertex Population 1_delayed, size: 1
Model: DelayExtensionVertex
constraint: PartitionerMaximumSizeConstraint(size=256)
constraint: PartitionerSameSizeAsVertexConstraint(vertex=Population 1)

Vertex Population 2_delayed, size: 1
Model: DelayExtensionVertex
constraint: PartitionerMaximumSizeConstraint(size=256)
constraint: PartitionerSameSizeAsVertexConstraint(vertex=Population 2)

*** Edges:
Edge Population 1_delayed_to_Population 0 from vertex: 'Population 1_delayed' (1 atoms) to
vertex: 'Population 0' (1 atoms)
  Model: No Model

Edge Population 2_delayed_to_Population 0 from vertex: 'Population 2_delayed' (1 atoms) to
vertex: 'Population 0' (1 atoms)
  Model: No Model

Edge Projection 1 from vertex: 'Population 2' (1 atoms) to vertex: 'Population 0' (1 atoms)
  Model: No Model

Edge Population 2_to_DelayExtension from vertex: 'Population 2' (1 atoms) to vertex: 'Population
2_delayed' (1 atoms)
  Model: No Model

Edge Projection 0 from vertex: 'Population 1' (1 atoms) to vertex: 'Population 0' (1 atoms)
  Model: No Model

Edge Population 1_to_DelayExtension from vertex: 'Population 1' (1 atoms) to vertex: 'Population
```

```
1_delayed' (1 atoms)
  Model: No Model
```

```
      Placement Information by Core
      =============================

Generated: Thu Dec  8 13:10:30 2016 for target machine 'virtual'

**** Chip: (0, 0)
Application cores: 15
  Processor 1: Vertex: 'Population 0', pop size: 1
            Slice on this core: 0:0 (1 atoms)
            Model: IFCondExp


  Processor 2: Vertex: 'Population 2_delayed', pop size: 1
            Slice on this core: 0:0 (1 atoms)
            Model: DelayExtensionVertex


  Processor 3: Vertex: 'Population 1_delayed', pop size: 1
            Slice on this core: 0:0 (1 atoms)
            Model: DelayExtensionVertex


  Processor 4: Vertex: 'Population 1', pop size: 1
            Slice on this core: 0:0 (1 atoms)
            Model: SpikeSourceArray


  Processor 5: Vertex: 'Population 2', pop size: 1
            Slice on this core: 0:0 (1 atoms)
            Model: SpikeSourceArray
```

tags.rpt

```
IP Tag on 127.0.0.1: tag=0 port=17896 ip_address=0.0.0.0 strip_sdp=True
```

```
      Placement Information by Vertex
 ===============================

Generated: Thu Dec  8 13:10:30 2016 for target machine 'virtual'

**** Vertex: 'Population 0'
Model: IFCondExp
Pop size: 1
```

Machine Vertices:
 Slice 0:0 (1 atoms) on core (0, 0, 1)

**** Vertex: 'Population 1'
Model: SpikeSourceArray
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms) on core (0, 0, 4)

**** Vertex: 'Population 1_delayed'
Model: DelayExtensionVertex
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms) on core (0, 0, 3)

**** Vertex: 'Population 2'
Model: SpikeSourceArray
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms) on core (0, 0, 5)

**** Vertex: 'Population 2_delayed'
Model: DelayExtensionVertex
Pop size: 1
Machine Vertices:
 Slice 0:0 (1 atoms) on core (0, 0, 2)

virtual_key_space_information_report.rpt

Vertex:
<spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex object at 0xa987cac>
edge:<spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex object at 0xa987cac>:Population 0:0:0, keys_and_masks:[KeyAndMask:0x4L:0xffffffffL]
edge:SimpleMachineEdge(pre_vertex=<spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex object at 0xa987cac>, post_vertex=Population 1_delayed:0:0, traffic_type=EdgeTrafficType.MULTICAST, label=machine_edge_forPopulation 1_to_DelayExtension, traffic_weight=1), keys_and_masks:[KeyAndMask:0x4L:0xffffffffL]

Vertex: Population 1_delayed:0:0
edge:SimpleMachineEdge(pre_vertex=Population 1_delayed:0:0, post_vertex=Population 0:0:0, traffic_type=EdgeTrafficType.MULTICAST, label=machine_edge_forPopulation 1_delayed_to_Population 0, traffic_weight=1), keys_and_masks:[KeyAndMask:0x0L:0xffffffffL]

Vertex:

<spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex object at 0xa987e4c>
edge:<spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex object at 0xa987e4c>:Population 0:0:0, keys_and_masks:[KeyAndMask:0x1L:0xffffffffL]
edge:SimpleMachineEdge(pre_vertex=<spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex object at 0xa987e4c>, post_vertex=Population 2_delayed:0:0, traffic_type=EdgeTrafficType.MULTICAST, label=machine_edge_forPopulation 2_to_DelayExtension, traffic_weight=1), keys_and_masks:[KeyAndMask:0x1L:0xffffffffL]

Vertex: Population 2_delayed:0:0
edge:SimpleMachineEdge(pre_vertex=Population 2_delayed:0:0, post_vertex=Population 0:0:0, traffic_type=EdgeTrafficType.MULTICAST, label=machine_edge_forPopulation 2_delayed_to_Population 0, traffic_weight=1), keys_and_masks:[KeyAndMask:0x2L:0xfffffffeL]