



Institut Supérieur d'Informatique, de
Modélisation et de leurs Applications

Campus des Cézeaux
1 rue de la Chébarde
TSA 60125
CS60026
63178 Aubière CEDEX

Laboratoire d'Informatique, Signaux et
Systèmes de Sophia Antipolis

I3S – UMR7271
UNS CNRS 2000
Route des lucioles
06900 Sophia Antipolis

Rapport d'ingénieur

Projet de 3^{ème} année

Filière : Génie Logiciel et Systèmes Informatiques

Utilisation de la plateforme Neuromorphic Computing Platform du Human Brain Project (HBP)

Présenté par Manon PREDHUMEAU et Justin GOUTEY

Responsable ISIMA

David HILL

Octobre 2016 - Mars 2017

Responsable professionnel

Alexandre MUZY

Projet de 120 heures

Remerciements

Nous tenons à remercier notre responsable professionnel, M. Muzy, qui a encadré notre projet depuis le laboratoire de recherche I3S à Sophia Antipolis. Nos remerciements vont également à notre responsable ISIMA, M. Hill, pour ses conseils et sa présence tout au long de ce projet. Nous remercions enfin M. Miramond, pour sa présence et son aide technique durant nos réunions en début de projet.

Résumé

Le **Human Brain Project** (HBP) de la Commission européenne vise à mettre en place une infrastructure de recherche pour une meilleure compréhension dans le domaine des neurosciences. Pour atteindre cet objectif, la communauté scientifique a accès à deux systèmes simulant le fonctionnement du cerveau: **SpiNNaker** et **BrainScaleS**. L'objectif de notre projet est de comparer ces **plateformes neuromorphiques** afin d'avoir un profil de chacune, référençant leur architecture, leurs performances et leur configurabilité. Nous avons fait des recherches approfondies sur les deux simulateurs pour comprendre leurs architectures matérielles et logicielles. Nous avons ensuite essayé de configurer les *émulateurs* disponibles pour tester des scripts **Python** décrivant des réseaux neuronaux sur nos ordinateurs personnels avant de les exécuter sur les plateformes distantes. Nous avons rencontré plusieurs problèmes : absence d'émulateur fonctionnel pour SpiNNaker, impossibilité de faire fonctionner correctement le simulateur BrainScaleS,... Nous avons réussi à exécuter un code représentant un réseau de neurones synfire chain sur SpiNNaker à partir de la plateforme web à disposition, mais nous n'avons pas accès à l'exécution détaillée. Les deux simulateurs semblent prometteurs, mais il est évident que SpiNNaker est à un stade supérieur de développement. Ainsi, nous nous sommes concentrés sur cette plateforme et avons obtenu des résultats intéressants, mais insuffisants pour analyser les performances.

Mots-clés : Human Brain Project, SpiNNaker, BrainScaleS, plateformes neuromorphiques, Python

Abstract

The **Human Brain Project** (HBP) is a European Commission flagship project and aims to implement a cutting-edge research infrastructure for a better understanding of brain and neuroscience. The HBP gives scientific community an access to two systems that emulate brain function: **SpiNNaker** and **BrainScaleS**. To make effective use of these systems, our project goal is to compare them to have a profile of each neuromorphic platform referencing their architecture, their performance and how configurable the system is. We have done some extensive research on both simulators to understand their hardware and software architectures. Then, we tried to setup available emulators to test some **Python** code on our personal laptops before running it on the real remote systems. We encountered several issues: there is no functional hardware emulator for SpiNNaker and we were unable to make the BrainScaleS simulator works properly. Finally, we managed to execute a code representing a synfire chain neural network on the SpiNNaker from the web platform, but we can not access the detailed execution. Both simulators seem to be promising but it is obvious that SpiNNaker is at a higher stage of development. Thus, we focused on this platform which gave us some interesting but insufficient results to analyze the performances.

Key words : Human Brain Project, SpiNNaker, BrainScaleS, neuromorphic platform, Python

Table des matières

Remerciements.....	ii
Résumé.....	iii
Abstract.....	iii
Table des matières.....	iv
Table des figures et illustrations.....	v
Introduction.....	1
I.Présentation du projet.....	2
1.Contexte du projet.....	2
1.1.Rappels sur les neurones biologiques.....	2
1.2.Le Human Brain Project.....	3
2. Sujet détaillé.....	4
3.Organisation mise en place.....	5
II.Comparaison des deux plateformes.....	7
1.Phase de documentation.....	7
1.1.BrainScaleS.....	7
1.Présentation.....	7
2.Les modules neuromorphiques.....	8
3.Utilisation du système.....	9
4.Logiciel de simulation du hardware.....	10
1.2.SpiNNaker.....	10
1.Quelques principes essentiels.....	11
2.Architecture physique.....	12
3.Couche applicative bas niveau.....	15
4.Couche applicative haut niveau.....	15
1.3.Un langage unique pour les deux simulateurs : PyNN.....	16
2.Tests de simulation du hardware.....	17
2.1.BrainscaleS.....	17
2.2.SpiNNaker.....	18
3.Recentrage sur Spinnaker.....	19
3.1.Le langage PyNN pour SpiNNaker : spyNNaker.....	19
3.2.Exécution en mode virtuel.....	23
III.Test de la plateforme et résultats obtenus.....	24
1.Création d'un compte et accès à la plateforme.....	24
2.Exécution d'un job.....	25
3.Résultats obtenus.....	27
Conclusion.....	30
Glossaire.....	viii
Table des annexes.....	x

Table des figures et illustrations

Figure 1: Transmission entre deux neurones.....	2
Figure 2: Diagramme de Gantt prévisionnel.....	5
Figure 3: Diagramme de Gantt réel.....	5
Figure 4: Le système BrainScaleS.....	7
Figure 5: Décomposition d'un module neuromorphique du système BrainScaleS.....	8
Figure 6: Modèle event driven appliqué à SpiNNaker.....	11
Figure 7: SpiNNaker en mars 2016.....	12
Figure 8: Liaisons bidirectionnelles entre les puces.....	13
Figure 9: Composition d'une puce SpiNNaker.....	13
Figure 10: Représentation schématique d'une puce.....	14
Figure 11: Couche applicative bas niveau.....	15
Figure 12: Partition et mapping des populations sur les coeurs.....	16
Figure 13: Exemple schématique de synfire chain.....	20
Figure 14: Demande d'accès "Test"	25
Figure 15: Formulaire de création d'un job.....	26
Figure 16: États des jobs.....	26
Figure 17: Détails d'un job terminé avec succès.....	27
Figure 18: Spikes générés par le code synfire chain.....	28

Introduction

Dans le cadre de la dernière année à l’Institut Supérieur d’Informatique, de Modélisation et de leurs Applications (ISIMA), un projet de 120 heures doit être réalisé par les étudiants. Sous la tutelle de M. Muzy, chargé de recherche au CNRS, ce projet est en lien avec le Human Brain Project de la Commission Européenne et s’inscrit dans le cadre de travaux de recherche concernant deux calculateurs neuromorphiques : SpiNNaker et BrainScaleS. L’objectif de notre projet est de comparer ces deux plateformes simulant le fonctionnement du cerveau humain en comprenant leurs architectures matérielles et logicielles et en testant leurs performances.

Le Human Brain Project (HBP) est un projet scientifique majeur initié par la Commission européenne en 2013. Ce projet rassemble des milliers de scientifiques et vise à mettre en place une infrastructure de recherche simulant le fonctionnement du cerveau humain pour une meilleure compréhension dans le domaine des neurosciences cognitives. Depuis fin mars 2016, le HBP met à disposition de la communauté scientifique un accès à six plateformes collaboratives. Ces plateformes, constituées de matériel physique, d’outils logiciels, d’interfaces de programmation, sont destinées à aider les scientifiques dans les domaines des neurosciences, de la médecine et de l’informatique. Dans le cadre de notre projet, nous avons utilisé à la plateforme « Neuromorphic Computing », qui donne accès à deux systèmes *neuromorphiques* : SpiNNaker et BrainScaleS.

L’objectif de notre projet est de comparer ces deux plateformes et de dresser un profil de chacune en comprenant leur architecture et en testant leurs performances et leur degré de configurabilité. Dans un premier temps, nous avons effectué une étude approfondie des deux simulateurs pour comprendre leurs architectures matérielles et logicielles. Nous avons dans un second temps tenté d’utiliser les émulateurs disponibles afin de tester des *codes de benchmarking* simples sur nos ordinateurs personnels. Nous avons alors rencontré de nombreux obstacles : SpiNNaker ne dispose actuellement pas d’émulateur totalement fonctionnel et nous n’avons pas réussi à faire fonctionner correctement le simulateur pour BrainScaleS. Nous avons donc procédé directement à partir de la plateforme web Neuromorphic Computing afin d’exécuter un premier code représentant un réseau de *neurones* synfire chain sur SpiNNaker. Malheureusement, nous n’avons pas accès à l’exécution détaillée, car la fonctionnalité est en cours de développement. Les résultats obtenus actuellement sont insuffisants pour analyser les performances de SpiNNaker.

Nous commencerons par présenter le contexte de ce projet, ses enjeux, ainsi que l’organisation que nous avons mise en place. Nous détaillerons ensuite la phase de comparaison des deux plateformes et de test des émulateurs. Enfin, nous verrons l’utilisation de la plateforme Neuromorphic Computing, le déroulement d’un *job* sur celle-ci et les résultats obtenus.

I. Présentation du projet

1. Contexte du projet

1.1. Rappels sur les neurones biologiques

On considère qu'aujourd'hui en Europe, les pathologies liées à un dysfonctionnement cérébral constituent 35 % des maladies, générant un coût de plusieurs centaines de milliards d'euros chaque année. Ceci n'est pas étonnant au vu de la complexité du cerveau et de notre manque de compréhension de son fonctionnement malgré les avancées dans ce domaine [1].

Le cerveau se compose de neurones, cellules nerveuses assurant la communication et le traitement d'informations. Ces neurones ont deux propriétés importantes: l'excitabilité, capacité de répondre aux stimulations et de les convertir en impulsions nerveuses et la conductivité, capacité de transmettre les impulsions. La transmission de ces impulsions est réalisée par les *synapses* (cf. Figure 1). Il s'agit d'une zone de contact reliant un neurone présynaptique à un neurone postsynaptique.

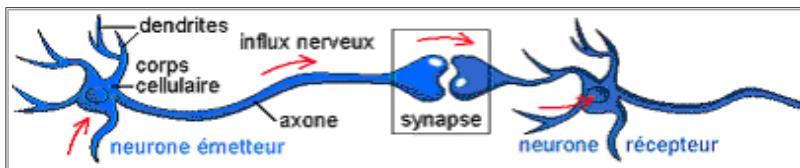


Figure 1: Transmission entre deux neurones

Source : <http://www.grand-dictionnaire.com/antispasmodiques.html>

Le nombre total de neurones dans un cerveau humain est estimé de 86 à 100 milliards [2] et il y a d'un à plus de 100 000 synapses par neurone (10 000 en moyenne). De plus, les réseaux formés par les neurones et les synapses sont évolutifs et se réorganisent tout au long de la vie, ce qui est appelé la *plasticité neuronale*.

L'ensemble de ces comportements fait du cerveau humain un organe extrêmement complexe et l'on peut se demander si sa modélisation est réalisable et si elle a vraiment un sens. La communauté scientifique rêve depuis bien longtemps de comprendre les liens entre nos raisonnements, nos comportements, nos émotions et les milliards de signaux générés par nos neurones, sans atteindre ce but malgré de nouvelles découvertes régulières.

1.2. Le Human Brain Project

C'est dans ce contexte que l'Europe et les États-Unis ont chacun mis en place un projet d'excellence dans le domaine des neurosciences. En 2013, la Commission Européenne a choisi le Human Brain Project (ou HBP) pour être l'une des deux Initiatives phare des Technologies Futures et Émergentes, soutenues par un milliard d'euros chacune, sur dix ans. Le HBP est prévu pour être réalisé en deux phases : la première de fin 2013 à mi-2016 et la seconde de mi-2016 à 2024 environ. Ce projet rassemble des milliers de scientifiques de 24 pays différents et a pour but la mise en place d'une infrastructure de recherche permettant de simuler le fonctionnement du cerveau humain pour une meilleure compréhension dans le domaine des neurosciences cognitives. Étant donnée la complexité du cerveau humain, le nombre de paramètres à explorer dans les simulations nécessite des processeurs hautement performants et une parallélisation massive des calculs.

Depuis fin mars 2016, le HBP est entré dans la seconde phase (phase opérationnelle) avec la mise à disposition pour la communauté scientifique d'un accès à six plateformes collaboratives de technologies de l'information et de la communication. Ces plateformes se composent de matériel de pointe, d'outils logiciels, de bases de données et d'interfaces de programmation [3]. Ce sont des plateformes collaboratives, les scientifiques sont appelés à les utiliser puis à partager leurs données et résultats pour les affiner et leur apporter de nouvelles fonctionnalités. Le but est d'aider les chercheurs à mettre en commun leur travail afin de progresser plus rapidement et plus efficacement en neurosciences, en médecine et en informatique.

Ces six plateformes HBP correspondent à :

- la plateforme neuro-informatique, pour le traitement de données en neurosciences,
- la plateforme de simulation du cerveau,
- la plateforme informatique haute performance, pour le calcul, l'analyse et le stockage de simulations complexes,
- la plateforme informatique médicale, orientée sur les données de patients réels et les maladies neurocérébrales,
- la plateforme informatique neuromorphique, pour l'accès à des calculateurs neuromorphiques.
- la plateforme Neurorobotics, qui permet de tester des modèles virtuels du cerveau en les connectant à des environnements de robots simulés.

Toutes les plateformes sont accessibles via un portail web : le *Collab HBP* [4]. Pour y accéder, les

utilisateurs doivent s'inscrire et ont ensuite accès à des tutoriels, de la documentation, un forum ainsi qu'au travail des autres utilisateurs s'il est public. Selon les plateformes, il est nécessaire de fournir des documents attestant son identité pour utiliser soi-même les matériels et logiciels mis à disposition. Ces plateformes sont en évolution constante, au fil des améliorations apportées par les développeurs. Un grand nombre de fonctionnalités est prévu ou en cours de développement.

2. Sujet détaillé

Dans le cadre de notre projet, nous nous sommes intéressés uniquement à la plateforme informatique neuromorphique (ou « Neuromorphique Computing Platform »). Celle-ci permet aux scientifiques de réaliser des simulations sur des systèmes configurables imitant le fonctionnement du cerveau. Actuellement, elle fournit un accès à quatre simulateurs neuromorphiques : BrainScaleS, SpiNNaker, BrainScaleS-ESS et Spikey. Nous nous sommes ici concentrés sur les deux premiers systèmes : SpiNNaker et BrainScaleS. BrainScaleS, également connu sous le nom de « modèle physique » de part son fonctionnement, est actuellement à Heidelberg en Allemagne. SpiNNaker, aussi appelé « Many Core », se situe à Manchester au Royaume-Uni [5]. Ces deux systèmes permettent de simuler des réseaux neuronaux à grande échelle avec des modèles simplifiés de neurones. Dans les deux cas, le point d'entrée pour modéliser ces réseaux est le langage *PyNN*, fondé sur Python.

L'objectif du projet est de comparer ces deux simulateurs en comprenant leurs architectures puis en testant leurs performances. La première étape consiste en des recherches approfondies sur les deux simulateurs pour comprendre leurs architectures matérielles et logicielles. Nous devons ensuite trouver des émulateurs disponibles et essayer de les configurer pour tester le code sur nos ordinateurs personnels avant de l'exécuter sur les vrais systèmes distants. Dans un second temps, le but est de définir des codes PyNN simples de benchmarking puis de les exécuter afin de comparer les résultats de performance. Ce projet doit permettre d'avoir un profil de chaque simulateur, référencant leur architecture, leurs performances et leur niveau de configurabilité.

Ce projet nous donne un accès privilégié à la plateforme HBP et un aperçu des technologies amenées à se développer dans les prochaines années. C'est un projet orienté recherche, qui est fortement en lien avec certains cours dispensés en 3^{ème} année à l'ISIMA et dans le Master Recherche Modèles, Systèmes et Imagerie (programmation parallèle, calcul haute performance, ...).

3. Organisation mise en place

Afin de bien visualiser les tâches, nous avons réalisé un diagramme de Gantt prévisionnel (cf. Figure 2) :

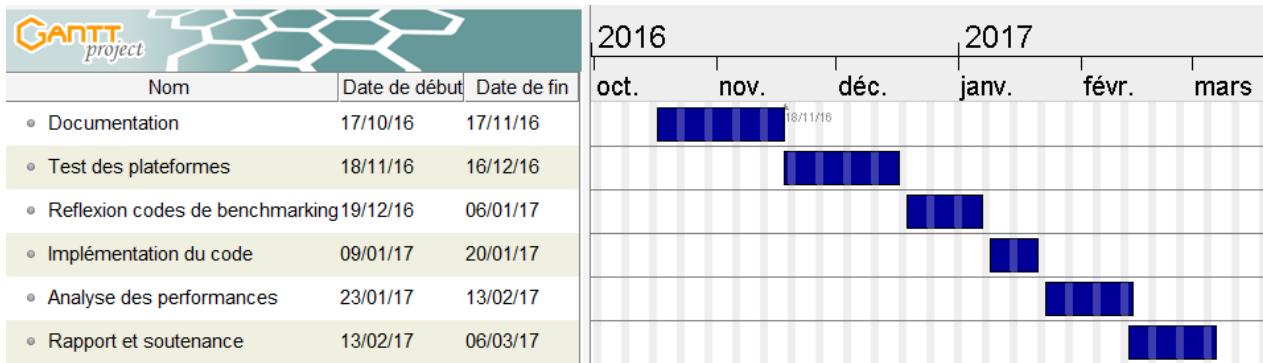


Figure 2: Diagramme de Gantt prévisionnel

Nous prévoyons un mois pour se documenter sur l'architecture des systèmes BrainScaleS et SpiNNaker. Nous avons prévu de répartir le travail : Justin s'occupe de BrainScaleS et Manon se renseigne sur SpiNNaker. Ensuite, un mois est prévu pour trouver des émulateurs, les installer sur nos machines personnelles puis les tester avec des codes PyNN simples. Nous considérons que les réflexions à mener sur les codes de benchmarking nécessitent deux semaines. Durant le mois de janvier, nous avons prévu d'implémenter les codes PyNN. Nous considérons que l'exécution des codes de benchmarking et l'analyse de performances des plateformes seront effectués de fin janvier à mi-février. Enfin, nous prévoyons un délai de trois semaines pour rédiger le rapport et pour préparer la soutenance.

Nous avons également réalisé un diagramme de Gantt réel pour comparer au précédent diagramme (cf. Figure 3) :

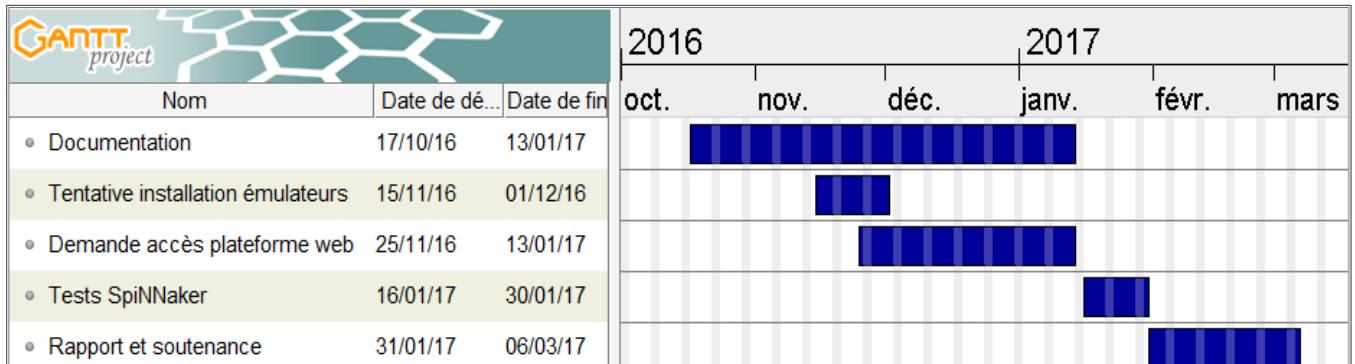


Figure 3: Diagramme de Gantt réel

Nous pouvons observer que la phase de documentation s'est étendue sur trois mois au lieu d'un mois. Ce changement est principalement dû au fait que nous n'avions pas prévu que la quantité de documentation disponible concernant SpiNNaker serait aussi importante. Le test des émulateurs a été plus rapide que prévu, car il n'existe pas d'émulateur fonctionnel pour SpiNNaker et nous n'avons pas réussi à utiliser celui de BrainScaleS.

Le projet a alors pris une nouvelle orientation. Au vu du peu de documentation disponible à propos de BrainScaleS et du manque d'émulateurs, nous nous sommes concentrés uniquement sur SpiNNaker. Ne pouvant pas tester les codes PyNN sur nos ordinateurs personnels, nous avons eu besoin de demander un accès à la plateforme Neuromorphic Computing. Ainsi, de fin-novembre à mi-janvier, au lieu de réfléchir à des codes de benchmarking et d'implémenter ces codes, nous avons poursuivi le travail de documentation sur SpiNNaker tout en essayant d'accéder à la plateforme web. La demande d'accès a pris beaucoup de temps, car la plateforme est sécurisée et il faut transmettre une preuve d'identité ainsi qu'un document assurant de l'usage que nous comptons faire de la plateforme. Étant donné que nous étions quatre futurs utilisateurs (M. Muzy, M. Miramond, Justin Goutey et Manon Predhumeau), la réunion et l'envoi de tous les documents nécessaires a pris un mois et demi. Nous avons ensuite pu tester un code de réseau neuronal simple sur le système SpiNNaker en utilisant la plateforme web. Nous avons alors rencontré un problème important : nous n'avons actuellement accès à aucun rapport d'exécution (excepté la trace d'exécution), il n'y a donc pas d'analyse de performances possible. Enfin, la rédaction du rapport et la préparation de la soutenance, prévues sur trois semaines, ont finalement débuté fin-janvier et ont été réalisées sur un mois.

Concernant le suivi du projet, nous envoyons un mail à nos référents, M. Muzy et M. Hill, environ une fois par semaine pour leur faire part de notre avancement sur le projet. En début de projet, nous organisions une visioréunion tous les dix jours environ pour faire un point, mais au vu de nos emplois du temps respectifs et de l'avancement dans le projet, nous avons choisi d'espacer les réunions et de mettre en place un suivi par mail. Nous rédigeons des compte-rendus régulièrement afin de résumer nos recherches de documentation et les différents articles lus. Cela nous a permis de rédiger au fur et à mesure des avancées et d'avoir du recul sur notre travail. Enfin, nous avons eu des contacts par mail avec les administrateurs des systèmes SpiNNaker et BrainScaleS et de la plateforme web afin de leur demander des précisions.

II. Comparaison des deux plateformes

L'objectif initial du projet est de comparer deux simulateurs neuromorphiques : BrainScaleS et SpiNNaker. Les principales caractéristiques de tels simulateurs sont la vitesse d'exécution, la possibilité de réaliser des simulations à grande échelle, l'élasticité (ils doivent pouvoir exécuter aussi bien de petites simulations avec peu de neurones que des simulations de plusieurs millions de neurones) et la faible consommation en énergie.

1. Phase de documentation

La première étape de ce projet a été de nous documenter sur les deux plateformes de simulation de réseaux de neurones biologiques : BrainScaleS et SpiNNaker.

1.1. BrainScaleS

Le système BrainScaleS de simulation de réseaux de neurones biologiques est un projet européen développé par 13 équipes de chercheurs et dirigé par une équipe basée à Heidelberg en Allemagne [6].

1. Présentation

Le but du projet BrainScaleS est de comprendre comment l'information est traitée par le cerveau et ce, à différentes échelles, c'est-à-dire à l'échelle du neurone, de la synapse ou carrément d'un cerveau fonctionnel. Le système BrainScaleS (cf. Figure 4) est une architecture physique de simulation de neurones biologiques. Concrètement, il se compose de 20 modules neuromorphiques, d'une infrastructure d'alimentation électrique et d'un cluster de calcul dédié à la gestion des modules et à l'établissement de l'environnement de simulation.



Figure 4: Le système BrainScaleS

Une simulation neuronale effectuée avec ce matériel évolue plus rapidement que l'équivalent biologique d'un ordre de grandeur de 10^3 à 10^5 selon le paramétrage du système. Ceci permet une simulation plus économique en énergie par rapport aux simulations de réseaux neuronaux classiques.

2. Les modules neuromorphiques

Chaque module neuromorphe constituant le système BrainScaleS se décompose comme sur la figure suivante (cf. Figure 5) :

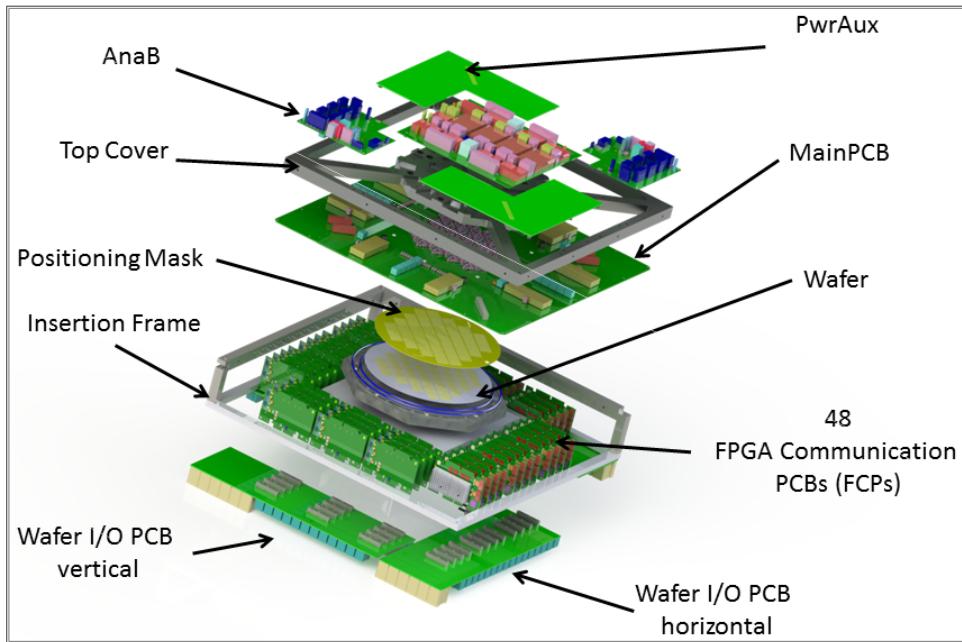


Figure 5: Décomposition d'un module neuromorphe du système BrainScaleS

Du point de vue de la simulation neuronale, les éléments intéressants sont le *wafer* et les 48 puces FPGA [7].

Le *wafer* est une plaque de silicium de 20 cm de diamètre qui contient les « neurones électroniques ». Il est composé de 384 puces hautement connectées entre elles. Ces puces de $5 \times 10 \text{ mm}^2$ sont appelées HICANN (High Input Count Analog Neural Network). Leur but est de simuler des neurones, par l'intermédiaire de signaux analogiques, et leurs synapses (communications et poids) par des signaux numériques. Chaque puce contient 128 000 synapses et 512 circuits « membrane de neurone » dont l'assemblage permet de former des réseaux de neurones. Chaque neurone peut être connecté avec jusqu'à 16 000 synapses.

Pour résumer, le *wafer* de chaque module neuromorphe est en mesure de contenir 196 608 neurones (384 puces x 512 membranes par puce) et 49 152 000 synapses (384 puces x 128 000 synapses par puce).

Il est cependant important de noter que ce modèle est limité par rapport à la réalité. En effet, dans un cerveau humain, un neurone peut être connecté, par l'intermédiaire des synapses, à entre un et

100 000 autres neurones contre un maximum de 16 000 dans le système BrainScaleS. Par ailleurs, le cerveau humain contient entre 86 et 100 milliards de neurones, ce qui serait l'équivalent de 10^5 modules neuromorphiques. De plus, dans le cas des puces HICANN présentes dans les wafers du projet BrainScaleS, si l'on souhaite utiliser le plus possible de synapses (16 000 par neurones au maximum à cause des limitations de connectivité du hardware), il n'est possible de simuler que 8 neurones par puce.

Les *cartes FPGA* sont présentes pour deux raisons. Tout d'abord, elles permettent d'établir la communication entre le réseau de neurones simulé sur le wafer et le cluster de calcul permettant de gérer la simulation. Ensuite, elles permettent le passage à l'échelle. En effet, ces cartes offrent la possibilité de créer des réseaux de neurones qui dépassent la taille limitée par le wafer en gérant les communications inter-wafer.

3. Utilisation du système

L'utilisation du système BrainScaleS se fait par l'intermédiaire de l'écriture de scripts Python et de l'*API PyNN*. Ces scripts permettent de paramétriser la simulation du hardware (vitesse, nombre de neurones, ...), mais également la quantité et le type de log en sortie de simulation.

L'utilisation de PyNN et de BrainScaleS dans un script Python se fait par l'intermédiaire de deux import :

```
import pyhmf as pynn
from pymarocco import PyMarocco
```

Le premier correspond à l'import d'une implémentation de PyNN tandis que le second permet de gérer l'utilisation du système BrainScaleS. Voici un exemple de script basique permettant de lancer une simulation avec un neurone pendant 10 ms :

```
import pyhmf as pynn                                // imports spécifiés précédemment
from pymarocco import PyMarocco

marocco = PyMarocco()                               // instanciation de pymarocco
marocco.backend = PyMarocco.ESS                     // simulation sur ESS (voir paragraphe suivant)
pynn.setup(marocco=marocco)                         // initialisation de PyNN

neuron = pynn.Population(1, pynn.IF_cond_exp, {})    // création de la population de neurones
```

```

pynn.run(10)                                // lancement de la simulation pendant 10ms
pynn.end()                                    // fin de simulation

```

Le paramétrage des logs de la simulation se fait via un autre import :

```

import pylogging
for domain in []:
    pylogging.set_loglevel(pylogging.get(domain), pylogging.LogLevel.ERROR)

```

En outre, il est possible de choisir la position des neurones dans les modules neuromorphiques avec l'aide d'un autre import de module :

```

import Coordinate as C
marocco.manual_placement.on_hicann(a_neuron, C.HICANNOOnWafer(C.X(5), C.Y(5)))

```

Le module Coordinate permet choisir sur quel « neurone électronique » du module neuromorphique placer le neurone « a_neuron » après qu'il ait été initialisé avec pynn.Population. De cette manière, il est possible de paramétrer au niveau hardware le positionnement de chaque neurone et donc de configurer le réseau selon notre volonté. De ce point de vue, la simulation sur système BrainScaleS est largement paramétrable par l'utilisateur.

4. Logiciel de simulation du hardware

ESS (Executable System Software) est un logiciel qui permet de simuler le système physique BrainScaleS dans le but de tester des petites simulations neuronales avant de passer à l'échelle sur le vrai hardware du système. Ce logiciel est programmé en C++/SystemC dans le but d'avoir un minimum de performance. Il peut être installé à partir des sources disponibles, après autorisation, sur le dépôt Git de la forge de l'université d'Heidelberg, ou à l'aide de Docker (dans un container Ubuntu 14).

Au niveau du fonctionnement, il s'utilise de la même manière que le système BrainScaleS hardware, c'est-à-dire par l'intermédiaire des scripts Python utilisant l'API PyNN.

1.2. SpiNNaker

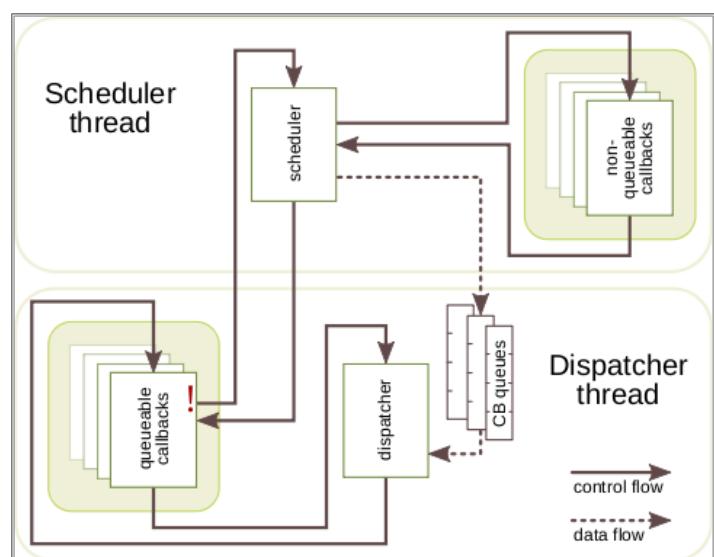
Nous allons à présent présenter le calculateur SpiNNaker (Spiking Neural Network Architecture). Cette plateforme est développée depuis 2005 par le groupe APT (Advanced Processor Technologies) de l'université de Manchester. Ce simulateur est basé sur trois couches : une couche matérielle, une couche applicative bas niveau et une couche applicative haut niveau.

1. Quelques principes essentiels

Dans le cas de SpiNNaker, les principaux axiomes spécifiques au calcul haute performance ne sont pas respectés : cohérence mémoire, synchronisation et déterminisme. Le système SpiNNaker repose sur plusieurs principes :

- L'incohérence mémoire : chaque processeur peut modifier la mémoire qui lui est accessible sans notifier ou se synchroniser avec les autres processeurs. Comme en biologie, seule l'information locale est connue. Les *cœurs* ont accès à deux mémoires : leur propre mémoire locale (fortement couplée) et la mémoire SDRAM de la puce. La mémoire locale au processeur n'est accessible que par lui-même et dans la SDRAM, on retrouve les données des synapses. 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 cœur accède à une plage d'adresses donnée. Ainsi, la SDRAM se comporte plus comme une extension de la mémoire locale que comme une mémoire partagée. Du point de vue du système, toute la mémoire est locale.
- Le non déterminisme : chaque puce est Globally Asynchronous, Locally Synchronous donc chaque cœur la composant est synchrone, utilise sa propre horloge et communique avec les autres blocs (cœurs) de façon asynchrone.
- Le simulateur fonctionne selon un modèle event driven (cf. Figure 6) : il n'y a pas de contrôle de l'exécution des fonctions mais des appels à base de callbacks. Les *spikes* fonctionnent comme des événements, envoyés par un neurone en spécifiant l'expéditeur mais pas le destinataire. On associe des appels de fonctions à des événements et à une priorité.

Quand un événement (arrivée d'un paquet par exemple) survient, la fonction correspondante est exécutée. Selon le type de callback (queueable ou non queueable), soit le scheduler exécute la fonction immédiatement et atomiquement, soit il la place dans une queue selon sa priorité. Quand le callback en cours se termine, le suivant dans la queue est exécuté.



L'exécution des callback en queue n'est pas forcément atomique. En effet, si un événement non queueable arrive pendant l'exécution d'un callback queueable, il est possible que celui-ci soit interrompu. Si la queue est vide, le dispatcher dort afin d'économiser de l'énergie et se réveille quand un événement arrive.

- La simulation se déroule en temps réel. Le but est de simuler 1 % d'un cerveau humain en temps réel, ce qui nécessite 50 000 puces environ. Cela explique le fait que l'on autorise la perte de certains paquets car cela se rapproche de la réalité dans laquelle certains signaux électriques ne peuvent pas toujours être transmis dans le cerveau.

2. Architecture physique

La machine SpiNNaker est aussi appelée machine « Many Core » car elle possède une architecture multi-cœurs dont le but à court terme est de simuler un milliard de neurones en temps réel (correspondant à environ 1 % d'un cerveau humain). Il existe plusieurs machines de type SpiNNaker [8] :

- la Jorg Conradt's avec une carte à une seule puce, pour les applications robotiques,
- la 4-nodes board avec 4 puces soit 72 cœurs, pour simuler des réseaux neuronaux à petite échelle, jusqu'à environ 10 000 neurones,
- la 48-nodes board avec 48 puces donc 864 processeurs, adaptée à des réseaux neuronaux jusqu'à quelques centaines de milliers de neurones.
- La 24-board frame intégrant 24 cartes de 48 nœuds, soit 20 736 processeurs. Elle permet de simuler des réseaux neuronaux jusqu'à plusieurs millions de neurones.



Figure 7: SpiNNaker en mars 2016

Dans le cadre du HBP, en mars 2016, la machine SpiNNaker utilisée (cf. Figure 7) comportait 600 cartes de 48 puces chacune. Cela correspond à 28 800 puces ou 518 400 cœurs, soit la moitié de l'objectif final (1 million de cœurs). La version actuelle est équivalente aux cerveaux de plusieurs souris. Chaque carte pouvant être utilisée seule, il est possible d'effectuer aussi bien une seule grande simulation mobilisant toutes les cartes que plusieurs petites simulations isolées et simultanées.

Chaque puce (aussi appelée nœud) est identifiée par une adresse fixe et communique avec les autres par 6 liaisons bidirectionnelles inter-puces qui permettent de former des réseaux de différentes topologies (cf. Figure 8).

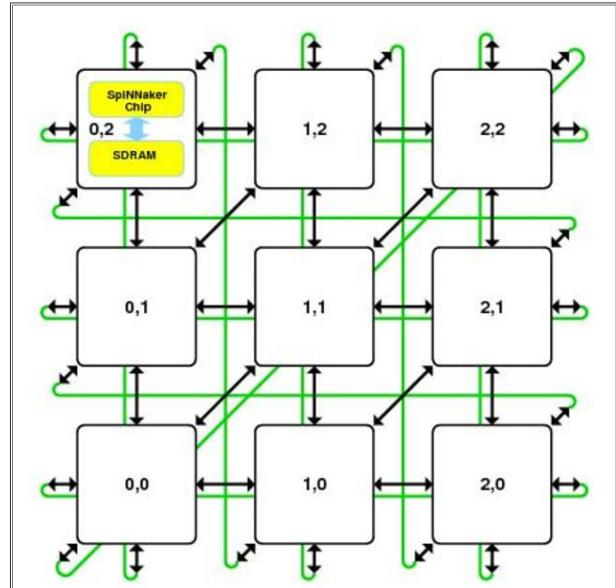


Figure 8: Liaisons bidirectionnelles entre les puces

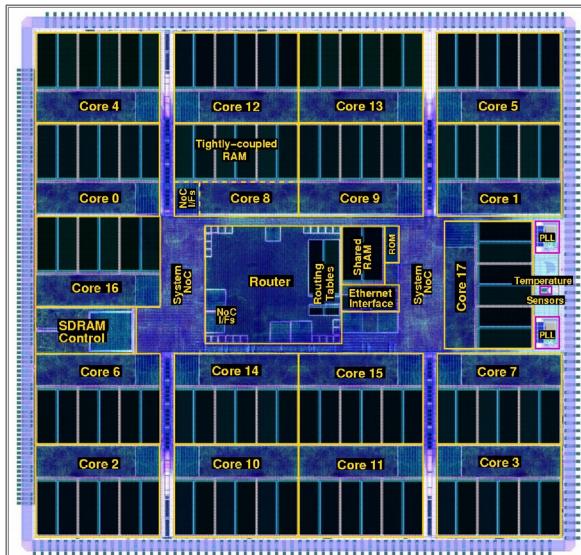


Figure 9: Composition d'une puce SpiNNaker

paramètres de la simulation.

Une seule puce, d'une surface de 10 mm sur 9,7 mm, peut simuler en théorie jusqu'à 16 000 neurones avec 8 millions de synapses, soit 1 000 neurones par cœur. En réalité, la couche logicielle impose une limitation à 256 neurones par cœur. Le système simule actuellement près d'un demi-milliard de neurones.

A l'intérieur d'une puce, chaque cœur ARM9 est repérable par ses coordonnées (x,y) et comporte un processeur avec 64 Ko de mémoire de données fortement couplée et 32 Ko de mémoire d'instruction fortement couplée. L'utilisation de cœurs ARM 968 permet de faire face à

une contrainte très importante pour les simulateurs de cette envergure : la consommation en énergie. En effet, si l'on veut utiliser plus d'un million de cœurs, il est absolument nécessaire que chaque cœur ait une faible empreinte énergétique, ce qui est le cas des processeurs ARM 968 (de 0,12 mW/MHz à 0,23 mW/MHz). Deux des inconvénients de ce type de processeur sont la mémoire limitée et le manque de représentation à virgule flottante. La pile logicielle sur la machine SpiNNaker utilise des nombres réels. Il existe une variante de la puce avec une unité à virgule flottante, mais cela aurait augmenté la consommation d'énergie [9].

Le système SpiNNaker est prévu pour consommer en dessous de 50kW en moyenne avec des pics entre 50kW et 100 kW. En comparaison, un cerveau humain moyen consomme environ 20 W [10]. Des estimations considèrent que le système final d'un million de cœurs consommera environ 90 kW [11]. Selon l'article « Power analysis of large scale real time neural networks on SpiNNaker » [12], la consommation sera de 1W par puce participant à la simulation et 360mW par puce au repos (soit 20mW par cœur au repos).

Au centre de chaque puce, un routeur multicast (cf . Figure 10) communique des paquets intra-chip

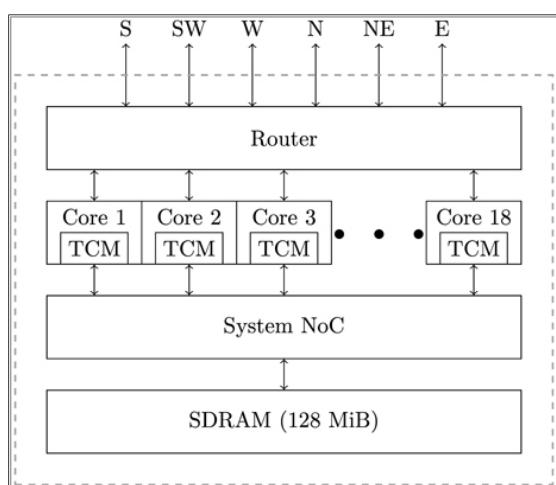


Figure 10: Représentation schématique d'une puce

et inter-chip de 40 bits (5 octets) ou de 72 bits (9 octets). La bande passante peut aller jusqu'à 5 milliards de paquets par seconde. Chaque paquet ne porte que des informations sur l'émetteur et le routeur est chargé de la distribution vers le(s) destinataire(s). On a ainsi un nombre très important de paquets de très petite taille en circulation, appelés « spikes ». Quand un neurone est simulé au delà d'un certain seuil, le spike est communiqué à tous les neurones voisins.

Chaque cœur a un contrôleur de communication qui génère et reçoit les paquets du routeur. Plusieurs routages sont possibles : multicast, point à point , au plus proche, route fixe, ...

Ce système possède cependant une limite, car même si la bande passante est très importante, des « goulots » (bottleneck) se forment lors de l'accès en écriture à la mémoire. Étant donné l'énorme taille du système et le nombre très important de composants, il est inévitable que certains

rencontrent des problèmes de panne et/ou de transmission des données. Pour gérer cela, une détection d'erreurs et un mécanisme de récupération des données sont intégrés au système. Il existe également un routage d'urgence, correspondant à une redirection temporaire des paquets sur les voisins en cas d'erreur ou de congestion.

3. Couche applicative bas niveau

Sur la couche matérielle vient s'ajouter une couche applicative bas niveau (cf. Figure 11) en deux parties : sur l'hôte (poste qui contrôle le système) et sur la puce.

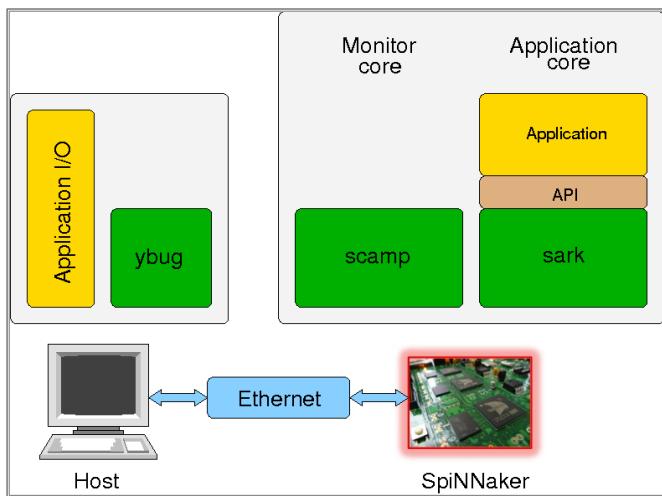


Figure 11: Couche applicative bas niveau

Sur l'hôte, on trouve un module permettant de gérer les entrées et les sorties de l'application ainsi qu'une application Ybug proposant une interface de commande et de debug.

Sur la puce, la couche applicative bas niveau est différente selon le type de cœur concerné.

Sur le cœur moniteur, une application nommée Scamp (SpiNNaker Control and Monitor Program) interagit avec Ybug sur l'hôte et Sark sur les cœurs applicatifs. Elle permet la communication entre les processeurs et la communication avec l'hôte via Ethernet. Grâce à Scamp, l'utilisateur peut collecter les résultats de la simulation (collecter les spikes et les récupérer par exemple). Les cœurs applicatifs utilisent Sark (SpiNNaker Application Run-time Kernel), qui exécute l'application et distribue puis planifie les callbacks des fonctions. S'il n'y a pas de tâches à exécuter, les cœurs passent en mode économie d'énergie.

4. Couche applicative haut niveau

Pour écrire des codes représentant des réseaux de neurones pour SpiNNaker, il faut utiliser le langage PyNN, dérivé du langage Python. L'implémentation de PyNN propre à SpiNNaker s'appelle *spyNNaker* et comprend quelques limitations spécifiques au simulateur.

SpiNNaker utilise l'application PACMAN (Partition and configuration management) qui partitionne automatiquement la *population* définie dans le script sur les cœurs (cf. Figure 12), de façon optimale selon les ressources disponibles et qui gère le routage. Le processus de mapping examine la définition du réseau neuronal et le partitionne pour le distribuer sur les différents cœurs. Il est possible d'influencer le partitionnement en ajoutant par exemple des contraintes pour limiter le nombre de neurones sur chaque cœur. Un algorithme de routage est ensuite exécuté pour calculer les routes empruntées par les paquets entre les cœurs. PACMAN possède plusieurs algorithmes de routage et il est possible de définir ses propres algorithmes [13].

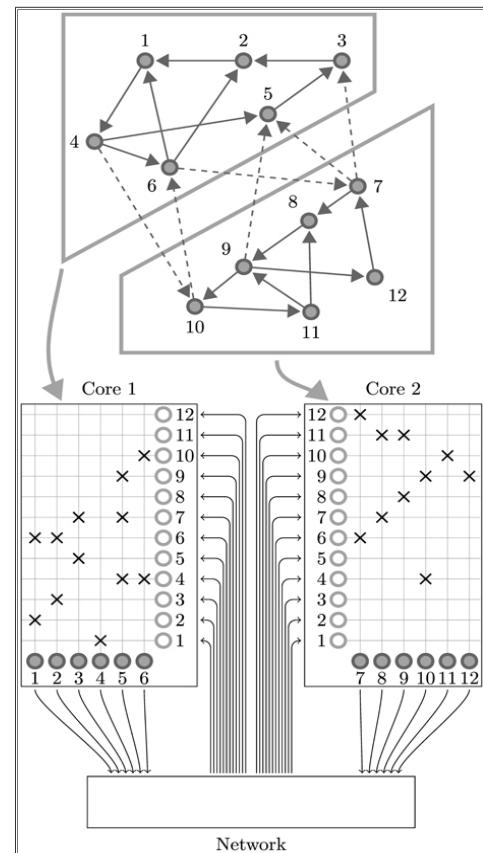


Figure 12: Partition et mapping des populations sur les cœurs

De nombreuses applications haut niveau (plus ou moins à jour) sont à disposition de l'utilisateur. L'application SpiNNmachine donne accès à une représentation de l'état actuel de la machine sur laquelle la simulation va être exécutée. SpiNNMan permet de communiquer avec SpiNNaker en envoyant des requêtes sur l'état de la machine (cœur/puces disponibles, cœur en utilisation ou non, boot correct,).

1.3. Un langage unique pour les deux simulateurs : PyNN

Modéliser des réseaux neuronaux est quelque chose de long et coûteux, le matériel est difficile à utiliser pour des non experts. C'est pourquoi, l'API PyNN est utilisée pour écrire les scripts Python correspondant aux réseaux neuronaux à simuler. PyNN est une API basée sur Python qui permet de décrire des modèles de réseaux neuronaux indépendants du simulateur utilisé. Pour cela, on utilise des populations de neurones et des projections. Cela permet d'avoir un haut niveau d'abstraction et de se concentrer sur les populations de neurones et les connexions en gardant accès aux neurones et synapses de façon individuelle.

PyNN fournit des modèles de neurones, synapses et connexions standards (donnant le même résultat sur différents simulateurs), qui seront ensuite traduits en noms spécifiques au simulateur effectivement utilisé. On écrit ainsi le modèle une fois et pour l'exécuter sur plusieurs simulateurs, il suffit de changer le nom du simulateur importé. Si on veut utiliser un seul simulateur, on peut utiliser les modèles de neurones et de synapses spécifiques à celui-ci (nativs) sans se restreindre aux standards. PyNN utilise la reconfigurabilité des cœurs pour exécuter des codes donnés par l'utilisateur.

Dans le langage PyNN, une « population » correspond à un groupe de neurones du même type. Les neurones sont liés par des « connecteurs », correspondant aux synapses. Un connecteur peut être de plusieurs types et modélise le retard, le poids et le comportement (plasticité) d'une synapse. Une « projection » contient toutes les connexions d'un certain type entre deux populations. Pour générer des signaux électriques, on utilise une « spike source », qui génère des spikes selon différents paramètres. La classe RandomDistribution permet de spécifier une distribution de nombres aléatoires pour la génération de spikes par exemple. On peut importer la bibliothèque NumpyRNG, qui correspond à un générateur Mersenne Twister ou NativeRNG selon le simulateur. Plusieurs fonctions permettent de contrôler la simulation : setup, run, qui avance la simulation d'un nombre choisi de millisecondes, reset, end, record, qui permet d'enregistrer les spikes, etc. Pour écrire un script PyNN fonctionnel sur plusieurs simulateurs, il faut commencer par importer les bibliothèques nécessaires. On doit ensuite définir la ou les population(s) de neurones ainsi que les propriétés des neurones et des synapses. On peut alors exécuter la simulation pour un temps fixe avec « run ». Si nécessaire on procède ensuite aux enregistrement des spikes, des populations, etc, avec la fonction « record ». Enfin, on récupère les enregistrements dans un fichier. On peut ensuite utiliser des outils pour analyser et visualiser les résultats.

2. Tests de simulation du hardware

Après la phase de documentation, nous avons tenté d'installer les émulateurs des deux systèmes afin de pouvoir tester le code PyNN sur nos ordinateurs personnels avant de l'exécuter sur les systèmes distants.

2.1. BrainscaleS

Nous avons tenté dans un premier temps d'effectuer quelques tests avec le container docker mis à disposition mais il s'est avéré que seuls les fichiers de test fournis semblaient vouloir s'exécuter. Nous nous sommes donc tournés vers une installation via les sources dans une machine

virtuelle Debian 8. Suite à plusieurs tentatives d'installation, il n'a pas été possible de faire fonctionner un script. En effet, l'interpréteur Python était incapable de localiser l'implémentation de PyNN nécessaire au fonctionnement de la simulation. Dans cette mesure, nous avons essayé de l'installer directement mais sans succès. Cet échec nous a fait reconsidérer l'intérêt de l'étude du système BrainScaleS. En effet, ce projet manque peut-être de maturité pour l'instant.

2.2. SpiNNaker

Nous avons commencé par chercher comment exécuter un code PyNN sur une machine SpiNNaker. D'après l'unique tutoriel trouvé [14], il est nécessaire d'installer spyNNaker pour exécuter un code PyNN sur SpiNNaker ou sur un émulateur. La première étape est d'installer les dépendances Python, puis d'installer spyNNaker et enfin de configurer le tout avec la commande « `python -c "import pyNN.spiNNaker"` », qui va créer un fichier de configuration “`.spynnaker.cfg`”. Dans ce fichier de configuration, il faut indiquer soit une machine SpiNNaker (hostname ou IP puis version), soit aucun SpiNNaker pour fonctionner en mode virtuel.

En recherchant de la documentation sur SpiNNaker, nous avons réalisé qu'un émulateur nommé « SpiNNakerEmulator » était disponible sur *GitHub* [15]. Le projet a été développé par l'équipe de Manchester et la dernière modification date d'octobre 2015. Ce projet fait partie d'un projet de plus grande envergure, rassemblant un ensemble de modules et bibliothèques utiles pour utiliser SpiNNaker. Après quelques recherches, il s'est avéré que malgré l'importante masse d'informations sur SpiNNaker, il existe très peu de tutoriels expliquant concrètement comment utiliser ces outils. Pour l'émulateur, il est annoncé qu'il « contient les fichiers qui répliquent la fonctionnalité de l'API (et qui ciblent le matériel ARM et SpiNNaker) et le rendent fonctionnel d'une certaine manière sur un poste de travail standard de type PC ».

Nous avons donc récupéré le code sur GitHub et nous l'avons compilé avec le Makefile à disposition. Dans un fichier de configuration inclus dans le projet (`default.cfg`), nous pouvons choisir le nombre de coeurs à simuler et la rapidité de la simulation (`x1, x10, ...`). Un script « `startEmulator.sh` » démarre l'émulateur qui se met alors en attente.

Dans le fichier de configuration “`.spynnaker.cfg`”, nous avons indiqué que la machine SpiNNaker à utiliser se trouvait à l'adresse 127.0.0.1. Dans un autre terminal, nous avons lancé un script d'exemple trouvé sur le site de PyNN [16], qui va communiquer avec l'émulateur, à l'adresse 127.0.0.1.

Nous avons alors rencontré un problème avec l'émulateur. Si nous précisons que l'émulateur

a 10x10 cœurs ou plus, cela entraîne une erreur de segmentation. Si l’émulateur a moins de cœurs, 5x5 cœurs par exemple, une erreur est affichée dans le terminal indiquant qu’il y a une erreur dans le code. Après de nombreuses recherches et tentatives de compréhension des erreurs sans succès, nous avons contacté les administrateurs sur le groupe Google de discussion concernant le SpiNNaker pour demander de l’aide. Quelques minutes plus tard, ils nous ont informé que la version actuelle de spyNNaker (3.0.0) n’était pas compatible avec l’émulateur. Les tutoriels sur le site ne sont donc pas tous à jour et il n’est indiqué nul part sur le projet GitHub que l’émulateur ne fonctionne plus avec la nouvelle version de spyNNaker. D’après les administrateurs, il faut soit avoir sa propre carte SpiNNaker, soit passer par le HBP Collaboratory et demander un accès à la plateforme pour exécuter des codes sur la machine SpiNNaker à Manchester.

3. Recentrage sur Spinnaker

Au vu des échecs d’installation des émulateurs, nous avons choisi de nous concentrer uniquement sur SpiNNaker. En effet, celui-ci paraît plus avancé dans son développement que BrainScaleS et propose un mode virtuel permettant de tester la validité des codes PyNN avant de les exécuter sur la machine de Manchester.

3.1. Le langage PyNN pour SpiNNaker : spyNNaker

Le langage PyNN adapté pour SpiNNaker s’appelle spyNNaker. En effet, SpiNNaker ne supporte pas tout ce qui est défini dans PyNN et pour n’utiliser que les fonctions et modèles supportés par la machine SpiNNaker, il est conseillé d’utiliser spyNNaker. Ce module possède cependant certaines limitations.

La version de PyNN utilisée n’est pas la plus récente (version 0.7 au lieu de la version 0.8) et toutes les fonctionnalités ne sont pas supportées : il manque certains modèles et certaines classes ne sont pas disponibles (PopulationView, Assembly,...). De plus, bien que le matériel impose une limite de 1000 neurones par cœur, les modèles neuronaux ont une limitation à 256 neurones par cœur et 256 synapses par neurone due à la façon dont les délais sont implémentés dans spyNNaker. Tous les modèles supportent des délais entre 1 timestep et 144 timesteps. Si le délai est supérieur à 16 time steps (soit 16 ms si 1 timestep vaut 1 ms), alors chaque atome aura un modèle de délai associé, ce qui prend un cœur sur la machine par délai extension, ce qui réduit le nombre de neurones simulable [17]. Les populations de plus de 256 neurones sont automatiquement fragmentées sur plusieurs cœurs.

Une fonctionnalité intéressante supportée par spyNNaker est la possibilité d’influencer

manuellement le partitionnement et le placement. Une contrainte de partitionnement peut être ajoutée à la population, ce qui peut limiter le nombre de neurones que chaque cœur contiendra au maximum avec l'instruction « `add_constraint(sim.PartitionerMaximumSizeConstraint(200))` ». Il est aussi possible de préciser la puce ou le cœur qui gérera la population. Avec « `population1.add_placement_constraint(x=1, y=1)` », la population de neurones « `population1` » sera placée sur le cœur de coordonnées (1,1). Il faut cependant faire attention à la taille de la population maximale supportée par ledit cœur, sous peine d'obtenir une erreur. Lorsque nous avons demandé aux administrateur de SpiNNaker s'il était possible d'utiliser ces contraintes manuelles sur SpiNNaker, ils nous ont affirmé que oui, mais qu'il était cependant important de se rappeler qu'il pourrait y avoir jusqu'à un milliard de neurones et 1000 milliards de synapses dans la simulation. Le placement manuel serait alors un exercice sans fin! Pour cette raison, il est recommandé d'utiliser les outils de placement automatique à disposition. Ils nous ont aussi rappelé que les puces n'étaient pas parfaitement homogènes, parfois une puce existe mais l'un des coeurs ne fonctionne plus et donc il pourrait y avoir un nombre différents de coeurs fonctionnels selon les puces.

Une fois les fonctionnalités et limitations de spyNNaker identifiées, nous nous sommes intéressés à un code exemple proposé sur le site de PyNN. Ce code modélise un réseau neuronal de type « *synfire chain* », c'est-à-dire une chaîne synchronisée de neurones avec une activité qui se propage dans le réseau (cf. Figure 13). La chaîne synfire est composée de petits groupes de neurones reliés entre eux dans une chaîne d'alimentation, de sorte qu'une vague d'activité peut se propager d'un groupe à l'autre dans la chaîne [18].

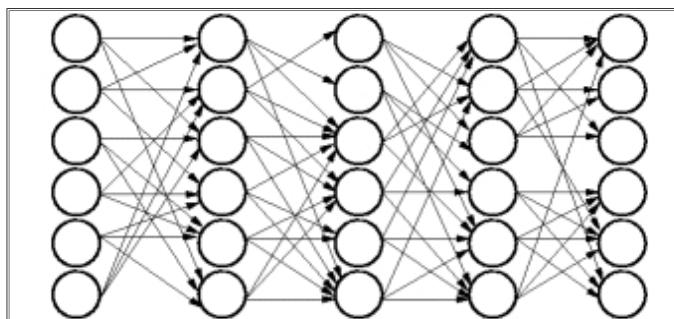


Figure 13: Exemple schématique de synfire chain

Source : [19]

Un exemple de code PyNN permettant de représenter un tel réseau est le suivant. On commence par importer les bibliothèques externes nécessaires, ici PyNN pour SpiNNaker, NumPy pour la génération de nombres aléatoires, et matplotlib pour le traçage de graphiques.

```

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

```

On définit les paramètres des neurones : nombre et taille des populations neuronales (ici 11 populations de 8 neurones), propriétés des neurones, poids synaptiques et retards, etc.

```

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

```

On initialise le simulateur :

```

sim.setup(timestep=0.1)

```

On crée ensuite 11 populations de neurones excitateurs « integrate and fire » (IF) et 11 populations de neurones IF inhibiteurs :

```

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)]

```

On crée dans un second temps les connexion entre chaque population excitatrice et la paire suivante de populations excitatrices et inhibitrices, et entre chaque population inhibitrice et la population excitatrice dans la même paire.

```

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')
```

La première paire de populations est stimulée par des spikes :

```
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')
```

On enregistre les spikes de toutes les populations :

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

On lance enfin l'exécution de la simulation :

```
sim.run(runtime)
```

Quand la simulation est terminée, on parcourt les populations, on récupère les spikes puis on trace un graphique (spike time en fonction de l'index des neurones) :

```
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")
```

Ici nous utilisons uniquement le simulateur SpiNNaker mais si l'on veut exécuter une même simulation sur plusieurs simulateurs différents, il est astucieux de ne modifier que le nom du simulateur PyNN à importer sans changer le code. Pour cela, il est possible de fournir le nom

du simulateur en argument de ligne de commande : `python run.py spinnaker`. Dans le code il faudra alors récupérer le nom passé en argument :

```
from pyNN.utility import get_script_args
simulator_name = get_script_args(1)[0]
exec("import pyNN.%s as sim" % simulator_name)
```

Il est aussi conseillé de séparer l'exécution et l'enregistrement des spikes (code soumis sur la plateforme) de l'analyse et de la visualisation graphique des résultats. Des outils, actuellement en développement, sont prévus sur la plateforme pour analyser des fichiers contenant les spikes enregistrés et produire des graphiques.

3.2. Exécution en mode virtuel

Une fois le code PyNN exemple analysé, nous avons tenté d'utiliser le mode virtuel proposé par l'équipe de Manchester [20]. Il est indiqué que ce mode permet aux personnes n'ayant pas accès à une carte SpiNNaker ou à la plateforme web de débugger les scripts.

Pour fonctionner en mode virtuel, il faut modifier le fichier de configuration « `.spynnaker.cfg` » et dans la section `[Machine]`, indiquer la taille de la machine SpiNNaker virtuelle en `x` et en `y`, soit le nombre de cœurs avec « `width=10` » et « `height=10` » par exemple pour une machine à 100 cœurs. Il faut spécifier que l'on est en mode virtuel avec « `virtual_board=True` ». Enfin, si l'on souhaite représenter une machine virtuelle câblée en toroïde (comme la machine SpiNNaker du HBP), il faut indiquer « `requires_wrap_around=True` ».

Lorsque la configuration est terminée, on peut exécuter des scripts PyNN à l'aide de la commande `python nomDuScript.py`. Le mode virtuel va alors générer une machine SpiNNaker avec les dimensions définies puis exécuter le mapping comme si c'était une machine réelle. Au moment de charger les données sur la machine, le mode virtuel s'arrête et affiche : "WARNING: The simulation is using a virtual machine and so has not truly ran, hence the list will be empty".

Le mode virtuel est un bon outil pour tester si le programme a des erreurs avant de l'exécuter sur la vraie machine mais ne permet pas d'obtenir les résultats d'exécution car celle-ci n'a pas lieu. Cependant, le mode virtuel génère une trace d'exécution ainsi qu'un ensemble de fichiers correspondant à la table de routage, la répartition des populations sur les cœurs, les spécifications du réseaux créé, la mémoire utilisée par cœur, etc (Annexes 1 à 4). Cet ensemble de fichiers peut être intéressant pour analyser les partitions et le placement automatique effectués.

III. Test de la plateforme et résultats obtenus

1. Crédit d'un compte et accès à la plateforme

L'interface web d'accès unique aux deux calculateurs BrainScale et SpiNNaker est la HBP Neuromorphic Computing Platform. Celle-ci donne la possibilité d'exécuter des codes représentant des réseaux neuronaux sur les systèmes neuromorphiques par la soumission de jobs. Il suffit de rejoindre la communauté HBP en demandant un compte puis de demander un accès aux ressources dont on a besoin, en confirmant son identité. Cela va entraîner la création d'un espace collaboratif mettant à disposition les outils nécessaires pour l'accès à la plateforme. On peut alors ajouter des membres et exécuter des simulations.

La création d'un compte requiert soit une invitation par un utilisateur possédant déjà un compte, soit une demande d'invitation en envoyant un court mail aux administrateurs de la plateforme expliquant l'intérêt que l'on porte à celle-ci. Benoît Miramond étant un collaborateur du projet, il nous a envoyé une invitation chacun, permettant la création de nos comptes. Il a aussi créé une « collab » publique, permettant de centraliser notre travail. Ces comptes nous donne accès à la plateforme sur laquelle nous pouvons consulter le forum et les tutoriels présents, demander des précisions aux administrateurs et avoir accès aux autres collabs publiques.

Cependant, cela n'est pas suffisant pour avoir la permission d'accéder aux systèmes neuromorphiques. Il est nécessaire d'estimer nos besoins puis de demander un accès selon ces besoins. Il existe trois types d'accès :

- accès « Test » : seul un résumé de l'utilisation que l'on souhaite faire de la plateforme est requis. Un quota de 5000 coeurs-heure est alloué ainsi qu'un espace de stockage temporaire de 1 Go.
- accès « Préparatoire » : un document expliquant l'utilisation souhaitée ainsi qu'une première expérience sur la plateforme sont requis.
- accès en mode « Projet » : pour des quotas plus importants, une page décrivant la motivation scientifique du projet ainsi qu'une demande des ressources nécessaires doivent être fournies.

Dans le cadre de ce projet, un accès « Test » est suffisant. Nous avons donc rédigé une requête (cf. Figure 14) spécifiant les besoins et le but du projet. Notre demande a été acceptée quelques jours plus tard.

The screenshot shows a software window titled "simple_neuron_Nice_1". At the top, there are two buttons: "Public" (unchecked) and "Member" (checked). Below the title bar, there are tabs for "Workspace" and "Resource Manager". On the right side of the window, there are icons for settings and closing the window. The main content area has a header "simple_neuron_Nice_1" and a message "Your request for access to the Neuromorphic Computing Platform is under review." Below this, there is a section titled "Abstract" containing the following text:

Dear Sir/Madam,

We are two french students (Justin Goutey and Manon Predhumeau) in last year of engineering school at ISIMA (Institut Supérieur d'Informatique, de Modélisation et de leurs Applications) in Clermont-Ferrand (France). As part of our final year, we have the opportunity to work on a 5-month team project. The project we have chosen consists of a study of two neuromorphic platforms: BrainScaleS and SpiNNaker. The aim is to compare these systems by testing their performance and understanding their architecture. In this context, we would like to have a test access to the SpiNNaker platform and to the BrainScaleS platform in order to test simple benchmarking codes. We are working with Mr. Alexandre MUZY and Mr. Benoit MIRAMOND, two HBP collaborators from the I3S laboratory in Sophia Antipolis (France).

Thank you for taking the time to consider this request.

Sincerely,

Manon Predhumeau

Figure 14: Demande d'accès Test"

Cela nous donne accès à 5000 cœurs par heure sur SpiNNaker et/ou 0.1 wafer par heure pour BrainScaleS.

Avant de pouvoir soumettre des jobs, il nous a fallu prouver notre identité. Pour cela, nous avons dû rassembler et envoyer les copies d'une pièce d'identité (carte d'identité ou passeport) de chaque collaborateur ainsi qu'un document « User agreement » par collaborateur complété et signé assurant que l'on compte faire un usage correct de la plateforme. Cela a pris beaucoup de temps car le « User agreement » nécessitait un tampon de la structure à laquelle nous appartenons et dans le cas de M. Miramond et de M. Muzy, il a fallu faire remonter le papier au CNRS.

2. Exécution d'un job

Dès que nous avons eu accès à la plateforme, nous avons pu réaliser les premiers tests. La demande d'exécution d'un code sur un des systèmes s'effectue en soumettant un job. Un job correspond à un script, des données optionnelles en entrée et une configuration du matériel.

Pour cela, il faut se rendre dans l'onglet « Job Manager » de la collab et cliquer sur « New job ».

Le formulaire ci-contre apparaît (cf. Figure 15). Il faut choisir la plateforme matérielle sur laquelle on souhaite exécuter le job (SpiNNaker, BrainScaleS, ...) puis entrer le code PyNN, l'URL d'un repository Git ou l'URL d'un zip contenant le code. On peut aussi spécifier une commande pour exécuter le code avec des arguments si nécessaire. Enfin, il est possible de préciser une configuration pour la plateforme ainsi que des fichiers d'entrée.

The screenshot shows a 'Create Job' dialog box. At the top, it says 'Hardware Platform' with a dropdown menu set to 'SpiNNaker'. Below that is a 'Code' input area containing PyNN code:

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

n_populations = 11
```

Below the code is a 'Command' input field, which is currently empty. Underneath is a 'Hardware Config' section, also currently empty. At the bottom left is a 'Cancel' button, and at the bottom right is a 'Submit' button.

Figure 15: Formulaire de création d'un job

Une fois le job soumis, il apparaît dans la liste des jobs (cf. Figure 16). On reçoit alors un mail indiquant que le job a été soumis et le job est marqué comme « submitted ». En cas de fonctionnement normal de la plateforme, c'est-à-dire à moins que la plateforme ne soit très utilisée, le job est exécuté en quelques minutes. On reçoit alors un nouveau mail indiquant le résultat de l'exécution (finished ou error). En cas d'erreur, on peut consulter un log indiquant la ligne fautive et le type d'erreur rencontré. En cas de succès, on peut consulter plus de détails sur l'exécution du job.

	ID	Status	Platform	Code	Submitted on	Submitted by
	91570	finished	SpiNNaker	import numpy.random import matplotlib.pyplot as plt...	2017-01-23 13:01:30	Manon Predhumeau
	91568	error	SpiNNaker	import numpy.random import matplotlib.pyplot as plt...	2017-01-23 12:53:34	Manon Predhumeau

Figure 16: États des jobs

3. Résultats obtenus

Nous avons exécuté avec succès un job sur SpiNNaker avec un code correspondant à un réseau neuronal de type synfire chain (cf. Annexe 5).

Nous pouvons voir sa date de soumission et sa date de fin ainsi que l'auteur de la soumission (cf. Figure 17). Si le job produit des fichiers de sortie, par exemple un graphique, nous pouvons le visualiser et le récupérer sur le stockage permanent de 1Go de la collab alloué lors de la demande d'accès (sinon il n'est plus consultable au bout de trois mois). Nous avons accès au code exécuté et à la configuration spécifiée (si aucune configuration n'a été renseignée, il est juste indiqué {"resource_allocation_id":98}).

/ Job 91570	
Status	finished
Submitted	2017-01-23 13:01:30
Completed	2017-01-23 13:04:05
Collab	simple_neuron_Nice_1
Platform	SpiNNaker

Figure 17: Détails d'un job terminé avec succès

Enfin, nous avons accès à une partie « Log » qui contient la trace d'exécution du job (version complète en Annexe 6).

A chaque étape de l'exécution, nous pouvons consulter le temps pris par la tâche. Le logiciel commence par prendre en compte la configuration du job indiquée (temps d'exécution, ...). Il génère une machine virtuelle et partitionne les populations de neurones sur celle-ci. Il se met ensuite en attente de ressources disponibles sur SpiNNaker jusqu'à trouver une carte libre :

```
2017-01-23 12:04:03 INFO: Found board with version [Version: SC&MP 3.0.1 at  
SpiNNaker:0:0:0 (built Wed Jul 20 10:07:23 2016)]  
2017-01-23 12:04:03 INFO: Detected a machine on ip address 10.2.225.105 which has 856  
cores and 120 links
```

Le logiciel place alors les populations de neurones sur les cœurs réservés et génère des rapports avec ces informations. Il crée ensuite les tables de routage, charge l'exécutable sur la machine et lance la simulation :

```
2017-01-23 12:04:07 INFO: *** Running simulation... ***  
Loading buffers (64 bytes)  
|0 50% 100%|  
=====  
2017-01-23 12:04:07 INFO: *** Awaiting for a response from an external source to state  
its ready for the simulation to start ***  
2017-01-23 12:04:07 INFO: Starting application (SCPSignal.SYNC0)  
2017-01-23 12:04:07 INFO: Checking that the application has started  
2017-01-23 12:04:07 INFO: *** Awaiting for a response from an external source to state  
its ready for the simulation to start ***  
2017-01-23 12:04:07 INFO: Application started - waiting 1.6 seconds for it to stop
```

```

2017-01-23 12:04:09 INFO: Application has run to completion
2017-01-23 12:04:09 INFO: Time 0:00:01.767999 taken by FrontEndCommonApplicationRunner
Getting spikes for Population 0
|0 50% 100%
=====
Getting spikes for Population 1
|0 50% 100%
=====
.....
Getting spikes for Population 20
|0 50% 100%
=====
Getting spikes for Population 21
|0 50% 100%
=====

```

Ici, on récupère les spikes de chacune des populations pour les afficher sur un graphique (cf. Figure 18) :

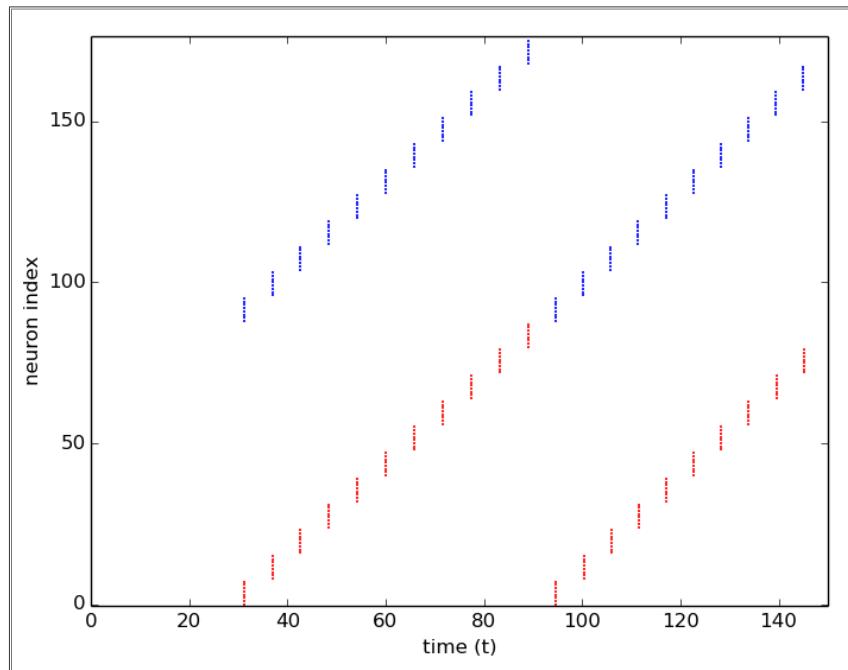


Figure 18: Spikes générés par le code synfire chain

Le code que nous avons exécuté est un exemple standard et nous avons pu confirmer que le graphique obtenu est conforme à ce qui était attendu. Comme prévu, l'activité générée par les spikes se propage de population en population.

Malgré l'accès à la trace d'exécution, nous n'avons aucun détail concernant le placement des populations sur les cœurs, les tables de routage, etc, comme c'était le cas lors de l'exécution en mode virtuel. Plus d'informations doivent être disponibles dans les rapports générés lors de l'exécution mais nous n'avons ici pas accès à ces rapports. Nous avons donc envoyé un mail aux

administrateurs de la plateforme pour demander plus d'informations. L'équipe de Manchester travaille actuellement sur la mise à disposition des données d'exécution mais n'a pas de date d'achèvement pour ce travail.

Nous avons ici atteint les limites du projet. Actuellement, les utilisateurs ont un accès très restreint aux machines neuromorphiques et nous ne disposons pas de suffisamment de données pour réaliser une étude de performances.

Conclusion

Le but du projet était de comparer deux plateformes simulant le fonctionnement du cerveau humain en comprenant leurs architectures matérielles et logicielles et en testant leurs performances. La recherche de documentation concernant les plateformes a été longue et fastidieuse car nous avons rencontré deux extrêmes. Il existe très peu d'informations concernant BrainScaleS alors que pour SpiNNaker une énorme masse d'articles et de documentation est accessible, bien que pas toujours à jour. Nous avons réussi à regrouper des informations concernant les deux plateformes et nous avons eu un aperçu des technologies de pointe actuelles dans ce domaine. Cela est valorisant pour nous d'avoir un accès privilégié à un projet d'envergure européenne. L'ensemble des articles étudiés pour SpiNNaker est disponible dans l'espace de stockage de la collab ainsi que dans un dossier Dropbox [21].

Au niveau de la comparaison des plateformes et de l'analyse de performances, nous avons rencontré plus de difficultés. Nous avons notamment mis du temps à essayer de faire fonctionner les émulateurs sans succès. En effet, l'émulateur disponible pour SpiNNaker n'est pas à jour et nous n'avons pas pu utiliser l'émulateur pour BrainScaleS, faute d'installation adéquate. Ces échecs nous ont amenés à réorienter le projet et à reconsidérer l'intérêt de l'étude du système BrainScaleS. Comparativement à SpiNNaker, il semblerait que ce projet manque de maturité pour l'instant.

Le second obstacle rencontré a été l'accès à la plateforme web Neuromorphic Computing. Nous avons mis du temps à obtenir un accès permettant d'exécuter des jobs sur les systèmes. Nous avons finalement pu exécuter un code représentant un réseau de neurones synfire chain et avons obtenu le graphique de sortie attendu. Malheureusement, nous n'avons actuellement pas accès aux détails de la simulation : placement des populations de neurones sur les cœurs, tables de routage, consommation en énergie, etc. Les informations accessibles concernant l'exécution ne sont pas suffisantes pour une analyse de performances. Un grand nombre de fonctionnalités sont annoncées mais peu d'entre elles ont été effectivement réalisées sur la plateforme actuelle.

Nous avons pu gérer un projet sensiblement différent de ce que nous étions habitués à réaliser. Ce projet s'inscrit dans un contexte de recherche et nous a permis d'avoir un réel aperçu du travail pouvant être demandé à un chercheur. En effet, nous avons dû nous informer sur les technologies actuelles et initier le travail de recherche en réalisant une importante mais nécessaire tâche de recherche bibliographique. Nous avons dû nous adapter au fil de l'avancée de nos recherches et réorienter le projet lorsque cela s'est avéré nécessaire.

Concernant l'avenir du projet, la question est d'avoir accès aux données d'exécution de nos

programmes PyNN pour analyser les performances du système. En passant par la plateforme web nous obtenons actuellement des données très limitées et nous ne connaissons pas la date de mise à disposition des rapports générés durant l'exécution. Une solution envisagée est de se procurer une carte SpiNNaker, qui permettrait d'avoir un accès direct à petite échelle. Nous pourrions ainsi développer nos propres fonctions et nous aurions plus de libertés. La prochaine étape sera donc de comparer les modèles de cartes existants et sera réalisée par Justin durant son stage de 5 mois dans l'équipe I3S à Sophia Antipolis.

Bibliographie

- 1: Bernard Esambert, Quelques mystères du cerveau, 3 octobre 2011
<http://parisinnovationreview.com/2011/10/03/mysteres-cerveau/>
- 2: Wikipédia, Neurone, <https://fr.wikipedia.org/wiki/Neurone>
- 3: Human brain project, <https://www.humanbrainproject.eu/platform-release>
- 4: Collab, <https://collab.humanbrainproject.eu/>
- 5: Documentation HBP, <https://electronicvisions.github.io/hbp-sp9-guidebook/index.html>
- 6: Site de BrainScaleS, <http://brainscales.kip.uni-heidelberg.de/>
- 7: Documentation BrainScaleS, <http://www.artificialbrains.com/brainscales>
- 8: Site de SpiNNaker, <http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/Access/>
- 9: https://studentnet.cs.manchester.ac.uk/resources/library/thesis_abstracts/MSc12/FullText/Moise-Mircea-fulltext.pdf
- 10: Docuention SpiNNaker, <http://www.artificialbrains.com/spinnaker>
- 11: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6515159>
- 12: Evangelos Stamatias, Francesco Galluppi, Cameron Patterson and Steve Furber, Power analysis of large-scale, real-time neuralnetworks on SpiNNaker
- 13: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4458614/>
- 14: Installation SpiNNaker,
<https://spinnakermanchester.github.io/spynnaker/3.0.0/PyNNOnSpinnakerInstall.html>
- 15: Émulateur SpiNNaker, <https://github.com/SpiNNakerManchester/SpiNNakerEmulator>
- 16: PyNN, <http://neuralensemble.org/PyNN/>
- 17: <https://spinnakermanchester.github.io/2015.005.Arbitrary/SpyNNakerLimitations.html>
- 18: M Herrmann, Analysis of synfire chains, 1995
- 19: Alexander Hanuschkin, Markus Diesmann, and Abigail Morrison, A reafferent and feed-forward model of song syntax generation in the Bengalese finch, 2011 Mars

20: Mode virtuel, https://spinnakermanchester.github.io/common_pages/3.0.0/VirtualMode.html

21: <https://www.dropbox.com/sh/2btqkkuy23ysuxj/AABJRmmd0Y3KwwD2all3lZrsa?dl=0>

Glossaire

API (Application Programming Interface) : ensemble normalisé de classes et de méthodes par laquelle un logiciel offre des services à d'autres logiciels.

ARM9 : famille de processeurs 32-bit reduced instruction set (RISC) produits par ARM Holdings.

BrainScaleS : système de simulation de réseaux de neurones biologiques basé à Heidelberg.

Carte FPGA (field-programmable gate array) : circuit logique pouvant être reprogrammé après sa fabrication.

Code de benchmarking : code permettant d'évaluer la performance relative d'un système.

Coeur (physique) : ensemble de circuits capables d'exécuter des programmes de façon autonome.

Collab HBP : portail web donnant accès aux plateformes collaboratives proposées dans le cadre du HBP.

Émulateur : reproduit le comportement d'un modèle ou d'un système dont toutes les variables sont connues.

ESS (Executable System Software) : logiciel qui permet de simuler le système physique BrainScaleS.

GitHub : service web d'hébergement et de gestion de développement de logiciels, basé sur le logiciel de gestion de versions Git.

Human Brain Project (HBP) : projet scientifique majeur initié par la Commission européenne en 2013.

Job : script, données d'entrée et configuration du matériel permettant une exécution sur un système distant.

Neuromorphique : système simulant le fonctionnement du cerveau.

Neurone : cellule nerveuse assurant la communication et le traitement d'informations.

Plasticité neuronale : fait que les réseaux formés par les neurones et les synapses sont évolutifs et se réorganisent tout au long de la vie.

Population : groupe de neurones du même type.

Puce : composant électronique.

PyNN : API basée sur Python qui permet de décrire des modèles de réseaux neuronaux indépendants du simulateur utilisé.

Python : langage de programmation.

Spike : paquet de données de très petite taille transmis entre les neurones simulés.

SpiNNaker : système de simulation de réseaux de neurones biologiques basé à Manchester.

SpyNNaker : implémentation de PyNN pour SpiNNaker.

Synapse : zone de contact reliant un neurone présynaptique à un neurone postsynaptique.

Synfire chain : chaîne synchronisée de neurones avec une activité qui se propage dans le réseau

Table de routage : table de correspondance entre l'adresse de la machine visée et le noeud suivant.

Wafer : plaque de silicium contenant les « neurones électroniques » dans BrainScaleS.

Table des annexes

Annexe 1 : Table de routage.....	x
Annexe 2 : Table de routage compressée.....	x
Annexe 3 : Utilisation de la mémoire par coeur.....	x
Annexe 4 : Informations de placement par coeur.....	x
Annexe 5 : Code PyNN d'un réseau neuronal de type synfire chain.....	xi
Annexe 6 : Trace d'exécution sur SpiNNaker obtenue par la plateforme web.....	xii

Annexe 1 : Table de routage

Router contains 4 entries

Index Key Mask Route Default [Cores][Links]

0	0x00000004	0xFFFFFFFF	0x00000280	False	[1, 3]	[]
1	0x00000002	0xFFFFFFF	0x00000080	False	[1]	[]
2	0x00000001	0xFFFFFFFF	0x00000180	False	[1, 2]	[]
3	0x00000000	0xFFFFFFFF	0x00000080	False	[1]	[]

0 Defaultable entries

Annexe 2 : 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	0xFFFFFFF	0x00000080	False	[1]	[]

0 Defaultable entries

Annexe 3 : Utilisation de la mémoire par 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

Annexe 4 : Informations de placement par cœur

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

Annexe 5 : Code PyNN d'un réseau neuronal de type synfire chain

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

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'][j],
                               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')

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")

```

Annexe 6 : Trace d'exécution sur SpiNNaker obtenue par la plateforme web

```

2017-01-23 12:03:47 INFO: Read config files: /home/spinnaker/.spynnaker.cfg,
spynnaker.cfg, /opt/git/sPyNNaker/spynnaker/spynnaker.cfg
2017-01-23 12:03:47 INFO: sPyNNaker (c) 2016 APT Group, University of Manchester
2017-01-23 12:03:47 INFO: Release version 3.0.0() - September 2016. Installed in folder
/opt/git/sPyNNaker
2017-01-23 12:03:47 INFO: Will search these locations for binaries:
/opt/git/SpINNFrontEndCommon/spinn_front_end_common/common_model_binaries :
/opt/git/sPyNNaker/spynnaker/pyNN/model_binaries
2017-01-23 12:03:47 INFO: Setting appID to 30.
2017-01-23 12:03:47 WARNING: A timestep was entered that has forced sPyNNaker to
automatically slow the simulation down from real time by a factor of 10.0. To remove
this automatic behaviour, please enter a timescaleFactor value in your .spynnaker.cfg
2017-01-23 12:03:47 INFO: Setting time scale factor to 10.0.
2017-01-23 12:03:47 INFO: Setting machine time step to 100.0 micro-seconds.
2017-01-23 12:03:47 INFO: Starting execution process
2017-01-23 12:03:49 INFO: Starting new HTTP connection (1): jabez.cs.man.ac.uk

```

```

2017-01-23 12:03:49 INFO: Time 0:00:00.061481 taken by
FrontEndCommonHBPMaxMachineGenerator
Generating a virtual machine
|0 50% 100%
=====
2017-01-23 12:03:49 INFO: Time 0:00:00.384202 taken by
FrontEndCommonVirtualMachineGenerator
Allocating virtual identifiers
|0 50% 100%
=====
2017-01-23 12:03:49 INFO: Time 0:00:00.015256 taken by MallocBasedChipIDAllocator
Partitioning graph vertices
|0 50% 100%
=====
Partitioning graph edges
|0 50% 100%
=====
2017-01-23 12:03:49 INFO: Time 0:00:00.047971 taken by PartitionAndPlacePartitioner
2017-01-23 12:03:49 INFO: Starting new HTTP connection (1): jabez.cs.man.ac.uk
2017-01-23 12:03:54 INFO: Time 0:00:05.039528 taken by FrontEndCommonHBPAllocator
2017-01-23 12:03:54 INFO: Starting new HTTP connection (1): jabez.cs.man.ac.uk
2017-01-23 12:03:55 INFO: Creating transceiver for 10.2.225.105
2017-01-23 12:03:55 INFO: Working out if machine is booted
2017-01-23 12:03:57 INFO: Attempting to boot machine
2017-01-23 12:04:01 INFO: Attempting to boot machine
2017-01-23 12:04:03 INFO: Found board with version [Version: SC&MP 3.0.1 at
SpinNaker:0:0:0 (built Wed Jul 20 10:07:23 2016)]
2017-01-23 12:04:03 INFO: Machine communication successful
2017-01-23 12:04:03 INFO: Detected a machine on ip address 10.2.225.105 which has 856
cores and 120 links
2017-01-23 12:04:03 INFO: Time 0:00:08.787303 taken by FrontEndCommonMachineGenerator
Allocating virtual identifiers
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.002366 taken by MallocBasedChipIDAllocator
Generating partitioner report
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.003131 taken by PartitionerReport
2017-01-23 12:04:03 INFO: Time 0:00:00.002081 taken by
FrontEndCommonApplicationGraphNetworkSpecificationReport
Filtering edges
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.010191 taken by GraphEdgeFilter
Placing graph vertices
|0 50% 100%
=====

=====
2017-01-23 12:04:03 INFO: Time 0:00:00.022417 taken by OneToOnePlacer
Generating placement report
|0 50% 100%
=====
Generating placement by core report
|0 50% 100%
=====
Generating SDRAM usage report
|0 50% 100%

```

```

=====
2017-01-23 12:04:03 INFO: Time 0:00:00.007935 taken by PlacerReportWithApplicationGraph
Routing
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.008062 taken by RigRoute
Allocating tags
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.004171 taken by BasicTagAllocator
Reporting Tags
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.001513 taken by TagReport
Getting the number of keys required by each edge using the application graph
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.009419 taken by FrontEndCommonEdgeToNKeysMapper
Allocating routing keys
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.046751 taken by MallocBasedRoutingInfoAllocator
Generating Routing info report
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.003030 taken by routingInfoReports
Generating routing tables
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.002452 taken by BasicRoutingTableGenerator
Generating Router table report
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.002024 taken by unCompressedRoutingTableReports
Compressing routing Tables
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.010309 taken by MundyRouterCompressor
Generating compressed router table report
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.001907 taken by compressedRoutingTableReports
Generating comparison of router table report
|0 50% 100%
=====
2017-01-23 12:04:03 INFO: Time 0:00:00.001248 taken by comparisonOfRoutingTablesReport
Generating sPyNNaker data specifications
|0 50% 100%
=====
2017-01-23 12:04:04 INFO: Time 0:00:00.500901 taken by SpynnakerDataSpecificationWriter
Loading routing data onto the machine
|0 50% 100%
=====
2017-01-23 12:04:04 INFO: Time 0:00:00.014264 taken by FrontEndCommonRoutingTableLoader
Clearing tags
|0 50% 100%
=====
Loading Tags
|0 50% 100%

```

```

=====
2017-01-23 12:04:04 INFO: Time 0:00:00.007574 taken by FrontEndCommonTagsLoader
Executing data specifications and loading data
|0 50% 100%
=====
2017-01-23 12:04:04 INFO: Time 0:00:00.240546 taken by
FrontEndCommonHostExecuteDataSpecification
Finding binaries
|0 50% 100%
=====
2017-01-23 12:04:04 INFO: Time 0:00:00.005139 taken by
FrontEndCommonGraphBinaryGatherer
Loading executables onto the machine
|0 50% 100%
2017-01-23 12:04:04 INFO: Starting new HTTP connection (1): jabez.cs.man.ac.uk
=====
2017-01-23 12:04:07 INFO: Time 0:00:02.765911 taken by
FrontEndCommonLoadExecutableImages
2017-01-23 12:04:07 INFO: Running for 1 steps for a total of 150.0 ms
2017-01-23 12:04:07 INFO: Run 1 of 1
Initialising buffers
|0 50% 100%
=====
2017-01-23 12:04:07 INFO: Time 0:00:00.037594 taken by
FrontEndCommonBufferManagerCreator
Updating run time
|0 50% 100%
=====
2017-01-23 12:04:07 INFO: Time 0:00:00.048147 taken by FrontEndCommonChipRuntimeUpdater
2017-01-23 12:04:07 INFO: Time 0:00:00.031850 taken by FrontEndCommonDatabaseInterface
2017-01-23 12:04:07 INFO: Time 0:00:00.018824 taken by
FrontEndCommonNotificationProtocol
2017-01-23 12:04:07 INFO: *** Running simulation... ***
Loading buffers (64 bytes)
|0 50% 100%
=====
2017-01-23 12:04:07 INFO: *** Awaiting for a response from an external source to state
its ready for the simulation to start ***
2017-01-23 12:04:07 INFO: Starting application (SCPSignal.SYNC0)
2017-01-23 12:04:07 INFO: Checking that the application has started
2017-01-23 12:04:07 INFO: *** Awaiting for a response from an external source to state
its ready for the simulation to start ***
2017-01-23 12:04:07 INFO: Application started - waiting 1.6 seconds for it to stop
2017-01-23 12:04:09 INFO: Application has run to completion
2017-01-23 12:04:09 INFO: Time 0:00:01.767999 taken by FrontEndCommonApplicationRunner
Getting spikes for Population 0
|0 50% 100%
=====
Getting spikes for Population 1
|0 50% 100%
=====
Getting spikes for Population 2
|0 50% 100%
=====
Getting spikes for Population 3
|0 50% 100%
=====
Getting spikes for Population 4
|0 50% 100%
=====
```

```

=====
Getting spikes for Population 5
|0          50%          100%|
=====
Getting spikes for Population 6
|0          50%          100%|
=====
Getting spikes for Population 7
|0          50%          100%|
=====
Getting spikes for Population 8
|0          50%          100%|
=====
Getting spikes for Population 9
|0          50%          100%|
=====
Getting spikes for Population 10
|0         50%          100%|
=====
Getting spikes for Population 11
|0          50%          100%|
=====
Getting spikes for Population 12
|0          50%          100%|
=====
Getting spikes for Population 13
|0          50%          100%|
=====
Getting spikes for Population 14
|0          50%          100%|
=====
Getting spikes for Population 15
|0          50%          100%|
=====
Getting spikes for Population 16
|0          50%          100%|
=====
Getting spikes for Population 17
|0          50%          100%|
=====
Getting spikes for Population 18
|0          50%          100%|
=====
Getting spikes for Population 19
|0          50%          100%|
=====
Getting spikes for Population 20
|0          50%          100%|
=====
Getting spikes for Population 21
|0          50%          100%|
=====
```