POLITECNICO DI TORINO
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

Modelizzazione energetica di tecniche di posizionamento di reti neurali su architettura neuromorfica multi-core

Relatori
Prof. Andrea Acquaviva
Dr. Gianvito Urgese
Dr. Francesco Barchi

Candidato
Adriano Lodia
matricola: 211292

Luglio 2018

Sommario

In questa tesi viene presentato uno studio sulla piattaforma neuromorfica SpiNNaker, acceleratore hardware in grado di simulare Spiking Neural Networks (SNN). Le SNN simulano il funzionamento del cervello tramite modelli matematici delle cellule cerebrali (neuroni) che comunicano tramite impulsi elettrici detti Spike. Per simulare le SNN. SpiNNaker posizione i neuroni sui processori di ogni suo chip, consentendo lo scambio di spike tramite pacchetti che viaggiano sulla sua rete di comunicazione inter-chip e intra-chip. In particolare, verranno studiate le fasi di partizionamento e posizionamento delle SNN e sull'impatto che queste fasi hanno sull'instradamento dei pacchetti e sul consumo energetico che ne deriva.

Questo lavoro di tesi verte sulle fasi di partizionamento e posizionamento dei neuroni e sull'impatto che queste fasi hanno sull'instradamento dei pacchetti e sul consumo energetico che ne deriva.

La prima fase del lavoro ha portato allo sviluppo di un framework capace di individuare l'effettivo percorso dei pacchetti (Spike) emessi da ogni neurone e alla costruzione di *grafi di propagazione*. Ogni chip possiede un Router che si occupa di generare l'instradamento dei pacchetti. Dall'analisi dei grafi di propagazione è emersa la presenza di destinazioni non previste dalla connettività della Spiking Neural Network. Durante la simulazione vengono, quindi, generati dei flussi di *pacchetti indesiderati* raggruppabili in due categorie: quelli inviati da un router ad un processore (R2C) e quelli scambiati tra due router (R2R).

La presenza dei pacchetti indesiderati è imputabile agli algoritmi usati durante le fasi di partizionamento, posizionamento ed indirizzamento della SNN su SpiNNaker. Utilizzando il calcolo dei pacchetti indesiderati ottenuto tramite il framework si è modellizzato il consumo energetico in surplus. I risultati evidenziano la presenza del 40% di energia usata per l'inoltro dei pacchetti indesiderati ottenuti durante la simulazione di una SNN usata come benchmark.

La seconda fase del lavoro si basa sull'evidenza che le tecniche usate per effettuare il partizionamento ed il posizionamento di una SNN sull'architettura siano la causa della presenza delle destinazioni indesiderate. Inoltre il posizionamento impatta sulla componente energetica R2R dovuta alla presenza di instradamenti più lunghi percorsi dagli spike emessi dai neuroni.

Il problema di partizionamento e posizionamento è stato quindi modellizzato

identificando possibili strategie per la sua risoluzione: Spectral Embedding, Scotch, Metis e Simulated Annealing sono gli algoritmi ed i programmi usati per trovare una migliore soluzione al problema. Le performance di posizionamento dipendono dalla elongazione sinaptica globale, cioè dalla distanza post-placement tra i neuroni che condividono una sinapsi. Usando una combinazione di partizionamento multilivello ed un posizionamento tramite simulated annealing è stato ottenuto un miglioramento del 28% dell'elongazione sinaptica globale ripetto alle tecniche attualmente implementate nella toolchain fornita con SpiNNaker. Questo comporta una riduzione energetica delle componenti di comunicazione R2R ed alla eliminazione delle destinazioni indesiderate.

Indice

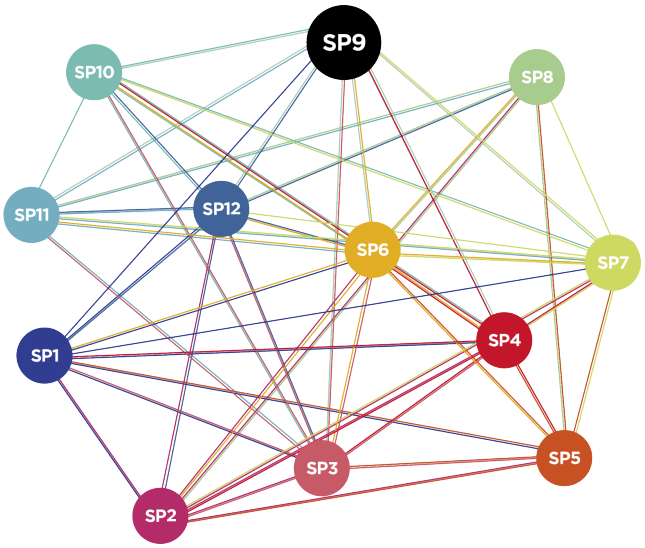

Figura 1.1: **Relazioni tra i SP**: Sono evidenziate le relazioni tra i Subprojects dello Human Brain Project. Fonte: humanbrainproject.eu

Capitolo 1

Introduzione

Lo Human Brain Project (HBP) è un progetto di ricerca ad ampio raggio finanziato dall'Unione Europea che si prefigge l'obiettivo di accelerare lo sviluppo scientifico nelle aree collegate allo studio del cervello. Questa accelerazione sarà raggiunta mettendo in atto una cooperazione strategica dei programmi di ricerca nelle neuroscienze, nella neuromedicina, nella neuroinformatica e nella progettazione hardware multiscala con la messa in opera di una grande infrastruttura di ricerca. I progetti inclusi nel programma sono 13 ed ognuno di essi collabora con gli altri (Figura 1.1) per raggiungere una sinergia utile al raggiungimento dell'avanzamento scientifico in questo campo di ricerca all'avanguardia.

Il lavoro discusso in questa tesi si colloca nel sottoprogetto 9 che prende il nome di "Neuromorphic Computing". L'obiettivo di questo progetto è la progettazine di

una nuova piattaforma hardware ispirata dalle reti cerebrali per superare i limiti fondamentali delle tecnologie convenzionali. La piattaforma neuromorfica fornirà l'accesso a due tipi di sistemi elettronici complementari. Il primo sarà implementato con l'ausilio di dispositivi analogici ultra veloci ed a basso impatto energetico che emulano i processi fisici del cervello; il secondo sarà basato su un alto numero si dispositivi digitali. Il presente lavoro si basa sul secondo modello: esso è basato sul progetto di un chip sviluppato dall'Università di Manchester in collaborazione on ARM ed il suo nome è SpiNNaker (Spiking Neural Network Architecture); SpiNNaker è un sistema completamente digitale che riunisce più di un milione di processori e può simulare 1,8 milioni di neuroni un enorme quantità di sinapsi attraverso una comunicazione multicast fra nodi. [19]

L'hardware di SpiNNaker è stato terminato ed è attualmente in produzione, ma il progetto del software è ancora in fase di sviluppo.

Il software ha il compito di assemblare e inviare al sistema i file di configurazione, a partire dalla descrizione delle reti di neuroni da simulare. É molto importate gestire in modo adeguato il traffico di pacchetti che vengono generati e scambiati dai processori dell'architettura, vista la mole di informazioni in gioco.

Il lavoro di tesi si è diviso in due parti:

- Creazione di un framework per l'analisi dell'impatto energetico derivanti dai pacchetti indesiderati creati dall'architettura
- Si sono testate delle tecniche di partizionamento e posizionamento delle reti di neuroni da simulare sul modello dell'architettura SpiNNaker

Nel sottoprogetto 9 vengono sviluppate due tipologie di Neuromorphic Computer Systems (NCS):

Neuromorphic Physical Model (NM-PM)

Progettazione iniziata con il "Fast Analog Computing with Emergent Transient States" (FACETS) [6] e parzialmente seguita durante il progetto "Brain-inspired multiscale computation in neuromorphic hybrid systems" (BrainScaleS).

Neuromorphic Many-Core (NM-MC)

Si tratta del progetto "Spiking Neural Network Architecture" (SpiNNaker).

Il FACETS rappresentato in Figura 1.2) è un sistema wafer-scale VLSI mixed-signal prodotto con tecnologia a 180nm, in cui i neuroni e le sinapsi sono simulati tramite ASIC analogici e comunicano attraverso potenziali d'azione binari in modo asincrono. Ogni neurone può utilizzare al massimo 16000 sinapsi e i pesi sinaptici e i parametri dei neuroni sono salvati su una memoria distribuita su tutta l'architettura. Il sistema contiene 200000 neuroni e 50000000 di sinapsi e può simulare un sistema biologico da fino a 100000 volte più velocemente del sistema stesso.

SpiNNaker è stato sviluppato dall'università di Manchester. La sua struttura completa prevede l'uso di 57000 nodi che includono 7TB di memoria e 1036800 processori ARM9 che comunicano tra di loro attraverso pacchetti di piccole dimensioni

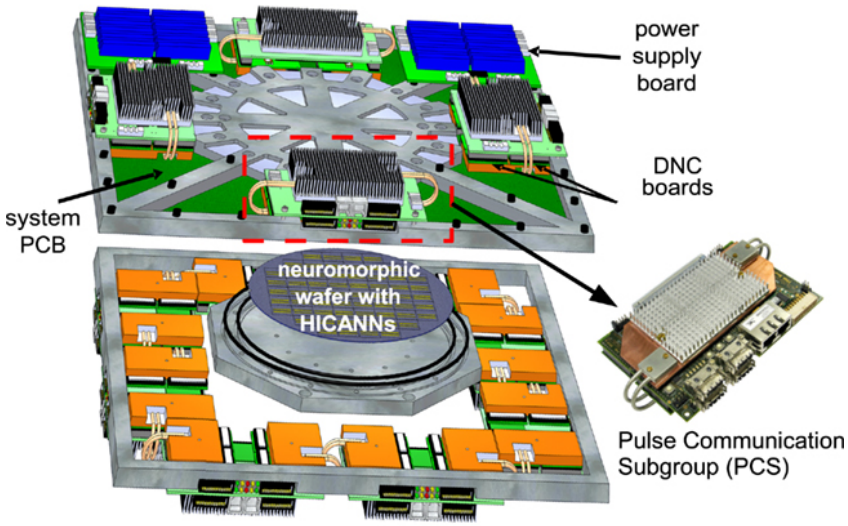

Figura 1.2: **Il NCS FACETS**: Fonte: facets-project.org

(40 o 72 bits) gestiti completamente in hardware. I nodi di questo sistema sono i chip SpiNNaker: ognuno di essi è un System in Package che contiene 18 processori ARM9, 128MB di memoria condivisa ed un Router. Ogni processore SpiNNaker è un'architettura Harvard e contiene una memoria per i dati da 64KB (DTCM) e una memoria per le istruzioni da 32KB (ITCM). Il blocco base dell'architettura è una Printed Circuit Board (PCB) che contiene 48 chip collegati in modo da formare una matrice esagonale 1.3. L'intero sistema che verrà assemblato entro il 2020 consiste in 1200 schede connesse tra loro per un consumo totale di 90kW di potenza [20]. In Figura 1.3) una foto del circuito stampato di SpiNNaker.

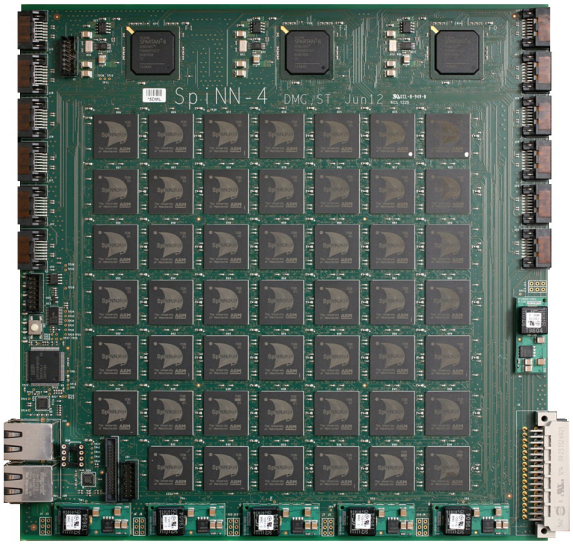

Figura 1.3: **Il NCS SpiNNaker**

1.1 Spiking Neural Networks

Descriverò soltanto le nozioni fondamentali per comprendere il funzionamento della coppia neurone-sinapsi e dei modelli matematici utilizzati per simulare questo il suo comportamento su NCS. Il cervello è formato da neuroni, cellule altamente specializzate che interagiscono fra loro per formare una rete di calcolo capace di risolvere problemi complessi [15].

In Figura 1.4 si vede la struttura di un neurone: i *dendriti* sono prolungamenti citoplasmatici che raccolgono i segnali provenienti da altri neuroni portandoli al *soma*, il corpo centrale della cellula neurale. La sua membrana cellulare ha una caratteristica tensione di membrana dovuta alle diverse concentrazioni ioniche; il potenziale cambia quando la cellula riceve stimoli da altri neuroni e quando raggiunge una tensione di soglia, la membrana si depolarizza e un potenziale d'azione chiamato spike si diffonde lungo l'assone, prolungamento atto a trasportare il segnale elettrico ad altri neuroni. Il collegamento tra *neurone efferente* (sorgente dello spike) e *neurone afferente* (obiettivo che riceve lo spike) è mediato tramite un segnale chimico tramite molecole dette neurotrasmettitori prodotte dalle sinapsi. Le sinapsi rilasciano i neurotrasmettitori quando vengono stimolate da uno spike; essi attraversano lo spazio sinaptico e i *recettori* presenti nel neurone afferente li ricevono. Il processo di trasformazione da segnale chimico a elettrico è un processo variabile nel tempo ed è un fenomeno detto *plasticità sinaptica*: è il principale responsabile della memoria e dell'apprendimento [25].

Le reti neurali artificiali di terza generazione Spiking Neural Networks (SNN) cercano di simulare il comportamento di un neurone, introducendo anche il fattore

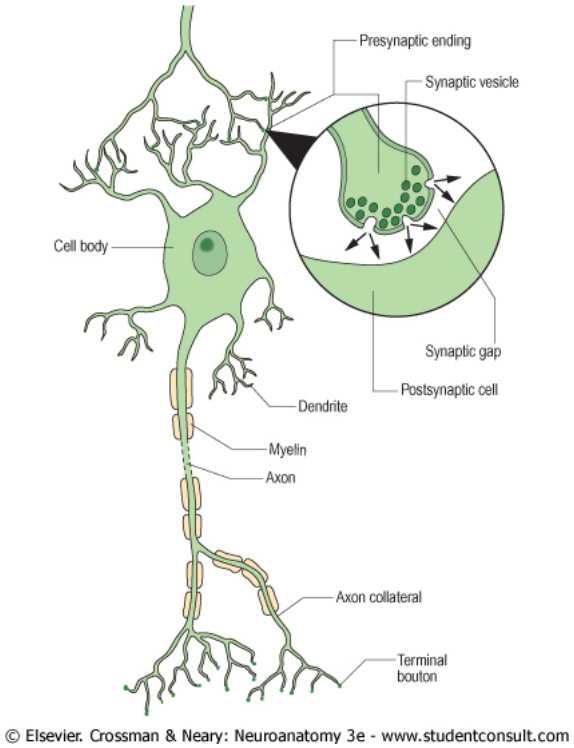

Figura 1.4: **Struttura di un neurone**

tempo continuo, cioè considerando anche la plasticità sinaptica. Il modello neuronale delle SNN simula correnti e tensioni di membrana e l'emissione degli spike.[10] Il modello Hodgkin-Huxley (HH) [13] è considerato il modello più dettagliato e verosimile di una cellula neuronale. Il modello HH descrive il neurone con un sistema di 12 equazioni differenziali e decine di parametri che specificano il circuito elettrico che modella il neurone. Per quanto questo modello sia verosimile al corrispettivo biologico la simulazione di una rete composta da modelli HH è computazionalmente onerosa. Perciò si preferisce usare due modelli più semplici e che necessitano di un carico computazionale minore: il Leaky Integrate and Fire (LIF) e il modello di Izhikevic (IZK).

Il LIF è una versione migliorata del modello Integrate and Fire (IF) basato sugli studi di Lapique [1] e [9] desritto in modo seguente:

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t)}{R_m} \rightarrow \tau \frac{dV_m(t)}{dt} = R_m I(t) - V_m(t) \tag{1.1}$$

$$se \quad V_m(t) \geq V_{th}, \quad allora \quad V(t) \leftarrow V_r \tag{1.2}$$

Il modello IZK presenta la stessa verosimiglianza biologica di HH e la stessa semplicità di LIF [14]. É un sistema di sole 2 equazioni differenziali e un sistema di

reset dopo lo spike simile al LIF:

$$\begin{cases} \frac{dV_m(t)}{dt} = 0.04V_m(t)^2 + 5V_m(t) + 140 - U(t) + I(t) \\ \frac{dU_m(t)}{dt} = a(bV_m(t) - U(t)) \end{cases} \tag{1.3}$$

$$se \quad V_m(t) \geq 30mV, \quad allora \begin{cases} V(t) \leftarrow c \\ U(t) \leftarrow U(t) + d \end{cases} \tag{1.4}$$

Su SpiNNaker sono presenti i modelli LIF e IZK sufficienti nella maggior parte degli ambiti applicativi.

1.2 SpiNNaker Project

Il progetto SpiNNaker si pone l'obiettivo di:

- fornire una piattaforma di calcolo ad elevato parallelismo e alte prestazioni per la simulazione in tempo reale di reti neurali ad impulsi su larga scala come strumento di ricerca per neuroscienziati ed esperti in informatica e robotica.
- Come ausilio nello studio di nuove architetture informatiche, che superino le regole del supercalcolo convenzionale, ma che porti a criteri fondamentalmente nuovi e vantaggiosi per il calcolo ad elevato parallelo e alta efficienza energetica.

L'architettura di SpiNNaker consiste di un array di processori ARM9 che comunicano utilizzando pacchetti di dati trasportati da una rete di interconessione ad hoc. Il sistema è asincrono, stocastico, a memoria condivisa. Data l'elevata impredittibilità e complessità del sistema sono stati integrati meccanismi di fault detection e recovery a diversi livelli di astrazione.

Di seguito descriverò l'architettura di SpiNNaker con un approccio top-down.

1.3 SpiNNaker Machine

In questa architettura ogni processore è un nodo. Ogni nodo può simulare fino a 1000 neuroni perciò caricare una rete da 10^8 neuroni sono necessari 1 milione di processori. Per raggiungere questo obiettivo il flusso di lavoro verrà svolto a step; ad ogni step l'architettura cresce nel numero di processori con una potenza di 10 che da il nome allo step di architettura:

- 10^2 **Machine** composta da 4 nodi e 72 processori ARM di cui 64 utilizzabili per la simulazione. É stata l'architettura di test, è un hardware 4 chip e la sua ultima versione porta il nome di Spin3

- 10^3 **Machine** è composta da 48 nodi per un totale di 864 processori ARM di cui 768 utilizzabili per la simulazione. É considerata il componente fondamentale per lo sviluppo di architetture più complesse, la sua implementazione è una scheda contenente 48 chip chiamata Spin5.

- 10^4 **Machine** è composta da 576 nodi per un totale di 10 368 processori ARM di cui 9 216 utilizzabili per la simulazione. Quest'architettura si compone di 12 Spin5 opportunamente collegate tra loro. Le 12 Spin5 sono alloggiate in un CardFrame capace di ospitare 24 schede.

- 10^5 **Machine** è composta da 5760 nodi per un totale di 103 680 processori ARM di cui 92 160 utilizzabili per la simulazione. Quest'architettura si compone di 5 CardFrame da 24 Spin5 l'uno (120 schede SpiNNaker48) disposti all'interno di un Rack.

- 10^6 **Machine** è composta da 57 600 nodi per un totale di 1 036 800 processori ARM di cui 921 600 utilizzabili per la simulazione. Quest'architettura è l'obiettivo finale del progetto, composta da 10 Rack (120 schede SpiNNaker48). Sotto l'ipotesi che ogni processore riesca a simulare 200 neuroni potrà simulare in totale circa $1{,}8 \cdot 10^8$.

1.4 Infrastruttura Comunicazione SpiNNaker

La Spin5 è il blocco base per l'architettura completa ed i suoi chip sono disposti in una matrice esagonale tramite connessioni fisiche, infatti ogni chip può comunicare con i 6 chip adiacenti attraverso link bidirezionali. Se 3 Spin5 vengono collegate tra di loro possono formate una struttura toroidale in modo da creare un'architettura chiusa, cioè in cui tutti i chip hanno 6 chip adiacenti. Le Spin5 si possono connettere tramite 3 FPGA poste ai bordi della board che si configurano per indirizzare correttamente i pacchetti da inviare. In Figura 1.5 si può vedere in maniera astratta l'archittettura di collegamento dei chip e delle FPGA.

I Chip invece comunicano tra loro scambiandosi pacchetti di piccole dimensioni (40 o 72 bit). Esistono quattro principali tipologie di pacchetti: Nearest Neighbor, Point to Point e Multicast. Ognuno offre una differente funzionalità e modalità di propagazione ma il loro utilizzo è limitato dal livello di Boot raggiunto dal sistema. L'architettura SpiNNaker ha tre livelli di avvio o Boot (Furber et al. 2014):

- 1. **Node Boot** All'accensione della scheda ogni chip elegge un core, tra i 18 disponibili, per svolgere il ruolo di Monitor Processor, gli altri svolgeranno il ruolo di Application Processor. Il Monitor Processor dovrà eseguire i test iniziali per confermare il buon funzionamento del Chip, gestire le comunicazioni con i chip vicini e configurare i componenti interni come memoria condivisa e

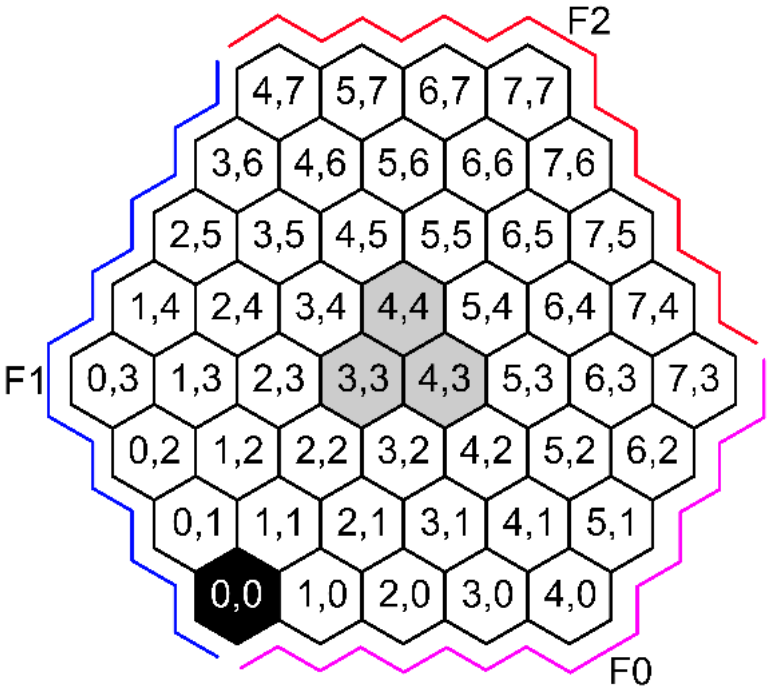

Figura 1.5: **La disposizione del chip SpiNNaker all'interno di una Spin5**

Router. Questa fase viene eseguita usando solo pacchetti Nearest Neighbor la cui gestione è da subito disponibile.

- 2. **System Boot** Quando il Monitor Processor di un Chip collegato all'interfaccia PHY riceve l'immagine del programma di sistema da eseguire lo esegue e propaga il programma ai suoi vicini che si comporteranno allo stesso modo finché tutti i Monitor Processor eseguiranno lo stesso codice. In questa fase ad ogni chip viene associato un ID a 16 bit (coordinate X, Y) in base alla distanza ed alla posizione rispetto al chip di partenza che assumerà ID (0,0). Al termine di questa fase viene abilitato lo scambio di pacchetti Point to Point.

- 3. **Application Load** Il Software Operativo da far eseguire agli Application Processor viene inviato ai Monitor Processor insieme ai dati di configurazione da immettere nella memoria del Chip, vengono caricate anche le Routing Table dei chip abilitando lo scambio dei pacchetti Multicast.

1.4.1 SpiNNaker Chip

Un chip SpiNNaker, i cui componenti sono raffigurati in Figura 1.6, è un System in Package che contiene:

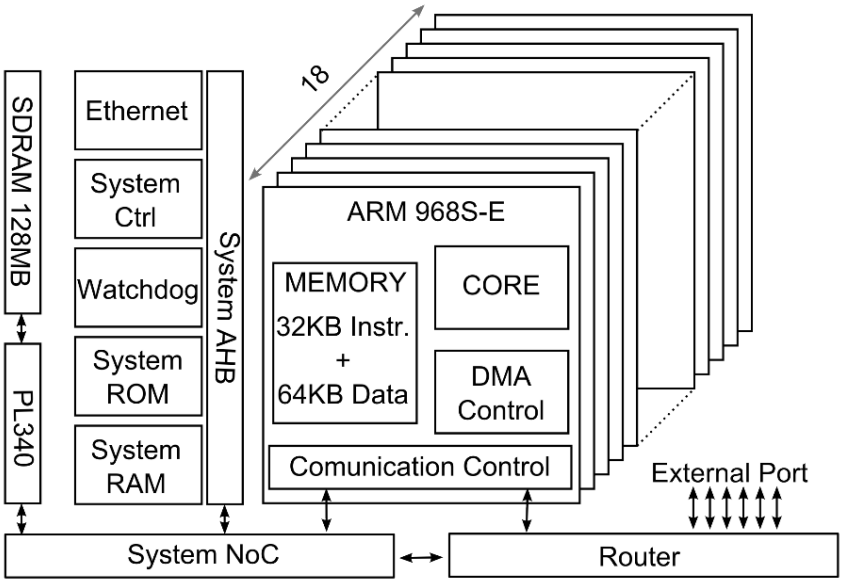

Figura 1.6: **Componenti interni al chip SpiNNaker [7]**

- 18 processori ARM968E-S
- Un Router
- 128MB di DDR SDRAM
- Una Network on Chip (NoC)
- System Controller con 32KB System ROM e 32KB System RAM
- Una memoria dati da 64 KB e una istruzioni 32 KB

I processori ARM non hanno unità floating point e le istruzioni e i dati viaggiano su un bus condiviso per realizzare una architettura Harvard. La frequenza del clock è 200MHz, condivisa da tutti i core dei chip. La DDR SDRAM da 128MB è inclusa nel chip package su un die dedicato. Uno spazio d'indirizzamento a 32 bit permette l'accesso alle 4 memorie disponibili e viene usato un indirizzamento virtuale per riferirsi ad un core dopo la fase di node boot. Il Monitor processor assume sempre l'indirizzo 0 e gli altri core i valori da 1 a 17. Uno dei core è lasciato come core *di riserva* perchè il processo produttivo può generare al massimo un core non funzionante per chip e, quindi i core utilizzabili dall'applicazione hanno id virtuale da 1 a 16. La Network On Chip interna può essere divisa in due principali zone: una usata per gestire le comunicazioni con il Router (Communication NoC) che ha input e output e una usata con i componenti interni (System NoC); quest'ultima permette di condividere tra i core una banda di 1GB/s verso la DDR SDRAM e fornisce un collegamento dedicato tra il Monitor Processor e le altre risorse di sistema.

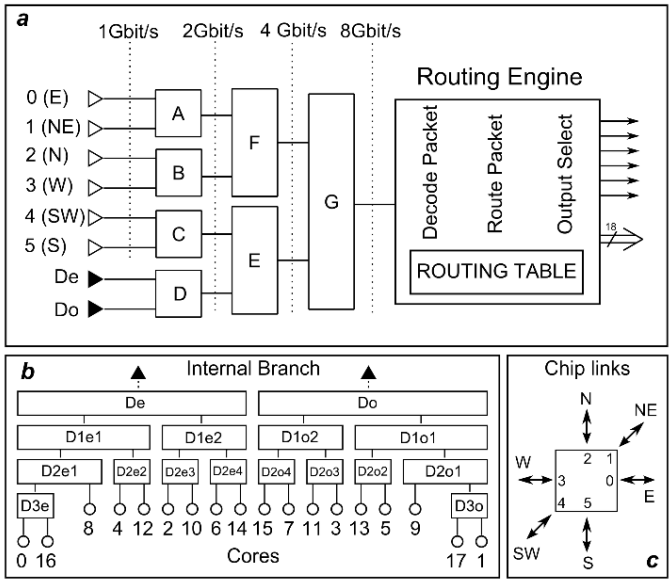

Figura 1.7: **Struttura del Router** [**7**]: a) Viene visualizzato l'albero di multiplexing in entrata nel Router ed il Router engine. b) Disposizione dei core ed i loro albero di mutiplexing prima di immettersi nell'albero principale. c) Direzioni delle porte di un chip.

1.4.2 Router

Il Router è integrato nel chip e permette di comunicare con i 6 chip adiacenti.

Ogni chip contiene un packet-switched router che può comunicare con tutti i 18 processori on-chip e i 6 router dei chip confinanti con esso tramite la Communication NoC. Questi 6 link ha due sotto-canali unidirezionali indipendenti, rispettivamente per l'input e l'output. Ogni canale unidirezione ha 8 linee di cui 7 usate per una trasmissione 2-of-7 ed una linea usata come ACK signal da parte del ricevente. Un pacchetto consiste in un byte di controllo ed una o due parole da 32 bit, quindi la dimensione può essere 40 o 32 bit. La struttura del router è raffigurata in Figura 1.7.

I pacchetti possono essere di 4 tipi ed alcuni di essi possono essere usati solo dopo alcune fasi di configurazione [18]:

- Nearest Neighbour (NN) in Figura 1.8 Dispoibili solo dopo il Node Boot, questi pacchetti possono essere scambiati solo dai Monitor Processor per testare la raggiungibilità reciproca. Non necessitano di regole di routing in quanto utilizzano solo i 6 chip fisici. Non verranno presi in considerazione in questa trattazione.

- Point to Point (PP) in Figura 1.9 Disponibili dopo la fase di System Boot in cui viene costruita la routing table da 2^16 righe da 3 bit (relativa ai PP). Con

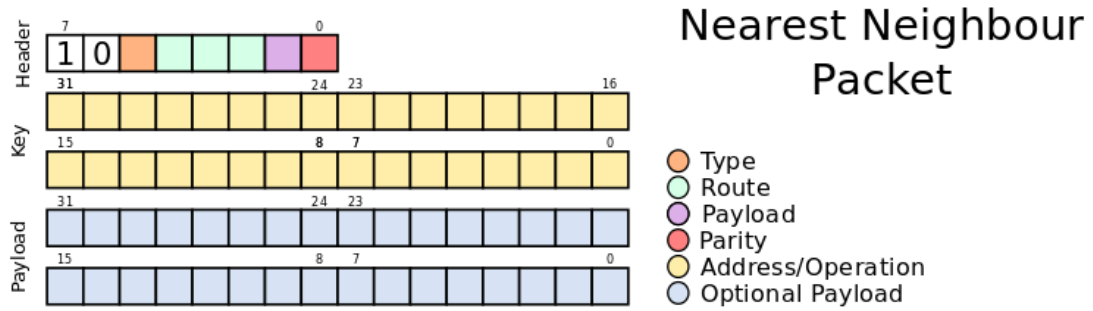

Figura 1.8: **Descrizione campi pacchetto Nearest Neighbours**

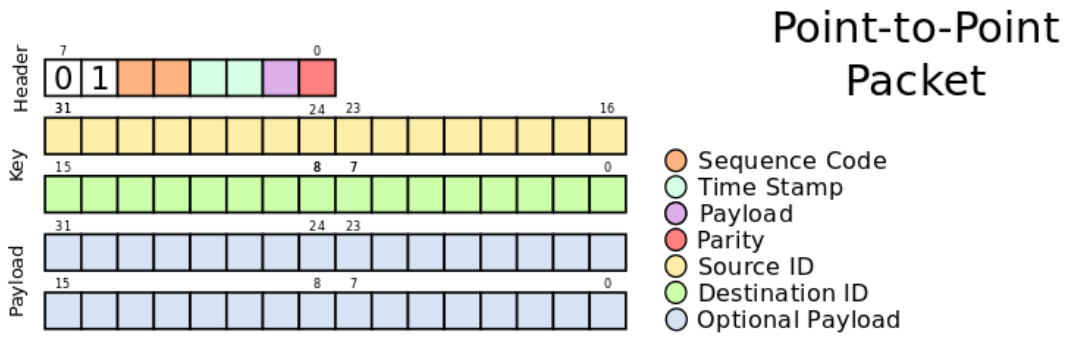

Figura 1.9: **Descrizione campi pacchetto Point-to-Point**

questo tipo di pacchetti si può raggiungere ognuno dei singoli nodi, identificati da un id a 16 bit, dell'architettura completa, avendo uno spazio di indirizzamento di 2^{16} (maggiore, quindi, del numero di nodi previsti dalla 10^6 Machine). Non verranno presi in considerazione in questa trattazione.

- Multicast (MC) in Figura 1.10 Disponibile solo dopo la fase di Application Load perchè dipende completamente dalle regole di routing caricate in questa fase. Questo perchè il pacchetto MC non contiene la destinazione, ma solo l'identificativo del neurone che ha scatenato l'evento. Il pacchetto viene duplicato per raggiungere tutti i destinatari. Viene utilizzato, quindi, per la comunicazione Core to Core.

- Fixed Route (FR) in Figura 1.11 Permette una trasmissione veloce Core to Core su route preconfigurate. É l'unico pacchetto con 64 bit di Payload.

Diamo uno sguardo in dettaglio ai pacchetti MC, unici attori della comunicazione in simulazione: ogni router dispone di una Content Addressable Memory (CAM) da 1024 celle, ognuna da 64 bit, contenenti le chiavi di Routing dei pacchetti MC da instradare e delle Maschere associate ad esse. Le Maschere servono per generalizzare

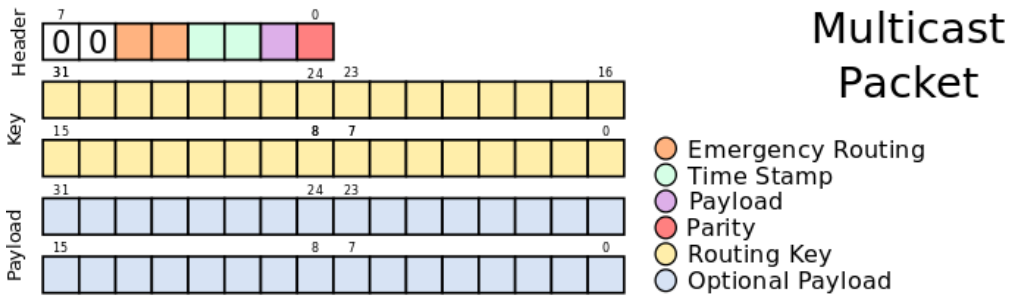

Figura 1.10: **Descrizione campi pacchetto Multicast**

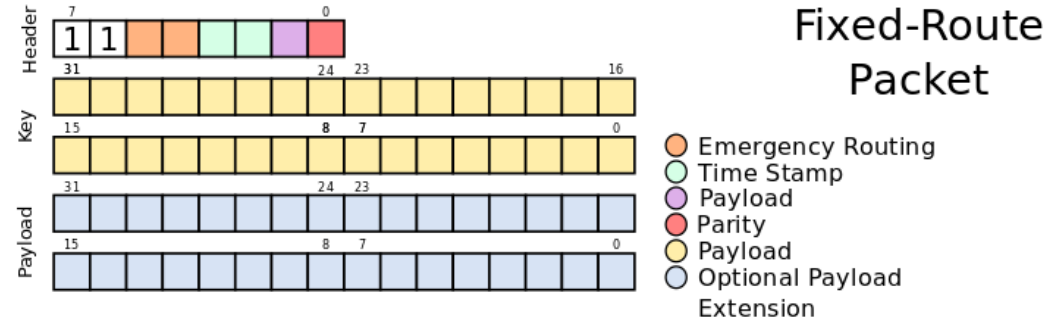

Figura 1.11: **Descrizione campi pacchetto Fixed Route**

una regola di routing: se la maschera ha uno dei bit a 1, quel valore della Routing Key è valido sempre, altrimenti per non invalidare il valore, la Routing key deve avere anch'esso il valore a 0. Quando un pacchetto MC arriva ad un router la sua SourceKey viene messa in AND con la maschera e confrontata con tutte le routingKey: se c'è match, la CAM restituisce un indirizzo della routing RAM in cui risiedono righe da 24 bit; i primi 6 bit indicano i link su cui duplicare il pacchetto, i restanti 18 bit indicano i core interni del chip. Se non c'è match si attiva il default routing ed il pacchetto viene duplicato sul link opposto a quello di arrivo. Le comunicazioni con l'host avvengono tramite pacchetti UDP/IP con un protocollo ad-hoc: l'host può comunicare solo con un Monitor Processor che poi gestisce la comunicazione on-board con gli altri chip utilizzando i pacchetti PP.

1.5 PyNN

Per utilizzare le SNN è necessario un'interfaccia per descriverle e simularle: per questi scopi la community NeuralEnsemble ha creato PyNN per supportarne la modellazione ad alto livello di astrazione. [3] PyNN è un package python simulator-indipendent che fornisce API per la descrizione della topologia di rete, per la scelta

dei modelli di neuroni desiderati e per la configurazione dei parametri.

PyNN fornisce una libreria di modelli standard di neurone, sinapsi e modelli per la plasticità sinaptica, che funzionano allo stesso modo sui diversi simulatori supportati. PyNN fornisce anche un insieme di algoritmi di connettività di uso comune (ad esempio one-to-one, casuale, dipendente dalla distanza, small-world), ma rende facile fornire la propria connettività in modo indipendente dal simulatore, usando il Set Connection Algebra [5] o scrivendo il codice Python ad-hoc. Per descrivere una rete neurale si devono dichiarare dei gruppi di neuroni che hanno gli stessi parametri, le *Popolazioni*. In seguito bisogna istanziare le *Proiezioni* tramite i connettori per rappresentare le sinapsi. PyNN genererà un grafo in cui le Popolazioni sono i nodi e le Proiezioni gli archi.

1.6 Partition and Configuration Manager for SpiNNaker

Dopo essere stata dichiarata con l'interfaccia PyNN, la rete deve essere posizionata su SpiNNaker. Per realizzare ciò è necessario partizionare, posizionare e riformare i collegamenti della rete. Questo lavoro è operato dal Partition and Configuration Manager for SpiNNaker (PACMAN) [**?**]. PACMAN si occupa di costruire il grafo *PartitionableGraph* partendo dalle popolazioni e dalle proiezioni fornite da PyNN; su questo grafo è possibile applicare le operazioni di partizionamento sui nodi ed archi detti *vertices* e *edges*. In uscita produce il grafo *PartitionedGraph* i cui nodi ed archi prendono il nome di *subvertices* e *subedges*. A partire da questo grafo possono essere applicati il posizionamento ed il routing.

Quando viene avviata la simulazione su SpiNNaker viene eseguito il System Boot della scheda, se risulta spenta, vengono avviate le fasi di partizionamento, posizionamento e routing (P2R) di PACMAN ed infine caricati i dati applicativi e le immagini dei programmi su ogni processore. Per configurare le operazioni P2R esiste un file di configurazione che PACMAN usa come input. Il buon funzionamento di PACMAN è essenziale per la riuscita della simulazione perchè i limiti fisici della scheda devono essere gestiti in modo da evitare hot-spot durante la simulazione o l'overload dei processori che devono simulare troppi neuroni o sinapsi attive. Perciò le popolazioni devono essere divise e posizionate correttamente.

Come configurazione in questo lavoro vengono scelti i seguenti algoritmi:

- **Partizionamento**: Partition and Place - Provvede a partizionare un *PartitionableGraph* in modo da ottenere subvertex con un numero di neuroni al massimo parti ai neuroni massimi simulabili all'interno di un processore, parametro impostato prima della simulazione e configurabile, tenendo conto anche della memoria SDRAM disponibile sui chip. **Posizionatore**: Radiale - Provvede a posizionare un *PartitionableGraph* con una disposizione circolare

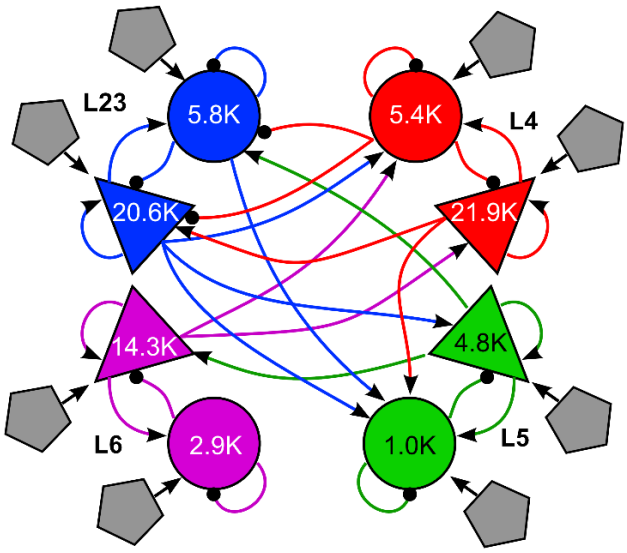

Figura 1.12: **Cortical Microcircuit** [7]. I quattro strati della corteccia cerebrale sono evidenziati con colori diversi, i triangoli sono popolazioni Eccitatorie, i cerchi sono le popolazioni Inibitorie. Per ogni popolazione è presente un pentagono di uguali dimensioni della popolazione a cui è connesso e rappresenta le popolazioni sorgenti.

intorno al nodo (0,0) **Routing**: Basic Dijkstra - Genera le regole di routing per ogni Subedge trovando il percorso più breve usando il Dijkstra Shortest Path Algorithm [2].

Il rischio principale degli algoritmi usati da PACMAN è non tenere in considerazione l'attività sinaptica e la connettività delle popolazioni. Questa tesi parte principalmente come analisi della bontà degli algoritmi di PACMAN in base all'attività sinaptica e propone nuovi algoritmi migliorativi che tengano in considerazione la connettività sinpatica.

1.7 Cortical Microcircuit

La SNN di test su questa trattazione è il Cortical Microcircuit (CM) [24]. Il CM rappresenta una sezione della corteccia cerebrale umana avente una base di are di $1mm^2$, composta da 4 layer: L2/3, L4, L5, L6 ognuno rappresentato da una popolazione eccitatoria e da una inibitoria Figura 1.12.

I neuroni della popolazione eccitatoria comunicano solo verso sinapsi con peso positivo, quelli delle popolazioni inibitorie solo verso sinapsi con peso negativo. La

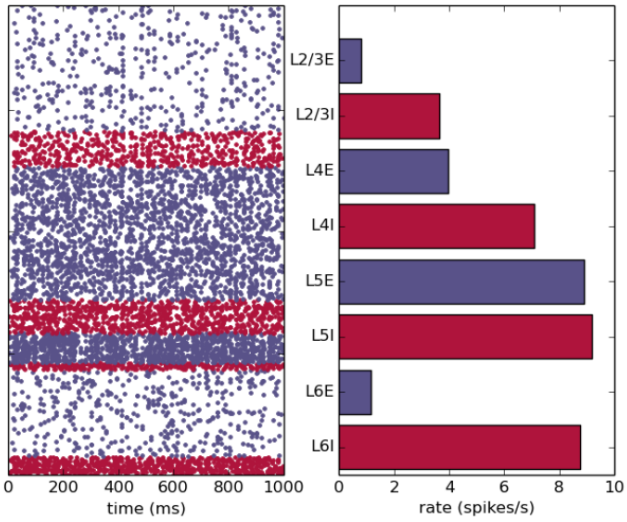

Figura 1.13: **Risultato della simulazione del CM al 15%**: il grafico è generato dal codice scritto da S. van Albada, M. Schmidt, J. Schücker ed A. Rowley.

rete è composta da 77000 neuroni e circa $3 \cdot 10^8$ sinapsi. Per simulare gli stimoli esterni alla rete istanziate delle popolazioni sorgente che emettono spike con un processo probabilistico di Poisson. I neuroni sono il doppio della rete originale perchè per ogni neurone della rete viene instanziato un neurone sorgente, ma le sinapsi aumentano solo di 77000 unità perchè un neurone sorgente connette solo un neurone della rete. Il risultato della simulazione è un tracciato di spike emessi dai neuroni dei vari layer rappresentato in Figura 1.13. I punti rossi sono gli spike emessi dalle popolazioni eccitatorie, quelli blu da quelle inibitorie. A destra un'istogrammadell'attività che rappresenta la media del numero di spike emessi al secondo.

Capitolo 2

Analisi dell'instradamento di SpiNNaker

Come spiegato nella sezione su PACMAN, ogni popolazione generata in PyNN può essere spezzata in più popolazioni parziali, se conta un numero di neuroni maggiore del massimo numero di neuroni simulabili da un processore; ne consegue che una popolazione di questo tipo è posizionata su più processori. Se non ci sono più processori disponibili su un chip, può capitare che una popolazione possa essere posizionata su due chip diversi. Dunque l'analisi parte dall'investigazione su come viene attuato il routing di PACMAN dopo il posizionamento delle popolazioni parziali. Per fare ciò si è utilizzato un test preliminare per capire effettivamente come funzionano le tecniche di routing nella versione presa in analisi (4.0.0) di SpiNNaker su una spin5.

2.1 Test Preliminare

Si è testato il routing su una rete semplice con poche popolazioni e pochi neuroni, così è stato possibile analizzare il comportamento del routing. Si è partiti, quindi, da una rete di test, la synfire chain [12]. Questa rete è una rete feed-forward con connessioni casuali tra i layer. Si è usata una versione semplificata di questa rete contenente soltalto 4 layer formati da un solo neurone ciascuno e 2 popolazioni contenenti 2 neuroni ciascuno, quindi le sinapsi non risultano più casuali perchè obbligate dal numero unitario di neuroni nel layer d'ingresso. Ogni popolazione è stata posizionata in un chip ed è stato fissato il limite di neuroni per processore ad uno: in questo modo se una popolazione possiede 2 neuroni PACMAN divide la popolazione in due popolazioni parziali. Abbiamo, quindi, 4 neuroni, ognuno appartenente ad una popolazione parziale, collegati in cascata da sinapsi e l'ultimo neurone è collegato al primo neurone. Infine una sorgente è stata collegata al primo layer, corrispondente al primo neurone come si vede in 2.1.

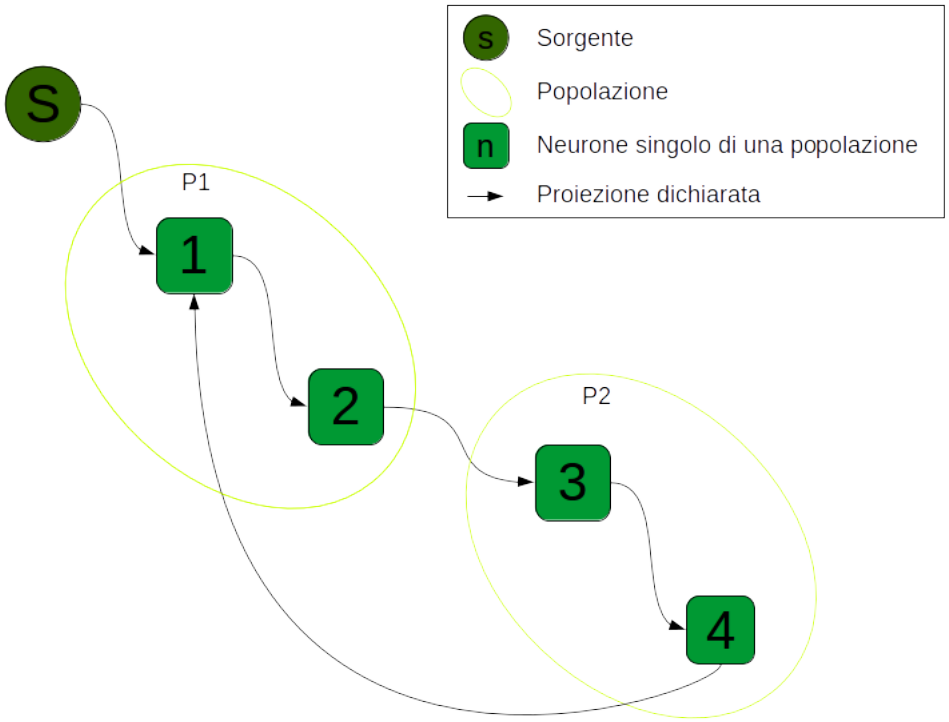

Figura 2.1: **Synfire Chain Test**: Schema della rete synfire usata come test preliminare.

In questo modo è molto facile capire se ci sono connessioni sinaptiche inaspettate perchè il path sinaptico dev'essere obbligato e ciclico. Perciò si è creata una rete così descritta utilizzando la libreria SpyNNaker, una libreria PyNN adattata per SpiNNaker, che include metodi per posizionare le popolazioni in chip e core stabiliti a priori. Poi, infatti, è bastato modificare la classe *transceiver* già presente nella libreria spinnman di SpiNNaker, connettersi alla spin5 e richiedere le routing table dei chip. A questo punto si sono confrontate le routing key delle routing table con gli identificativi dei neuroni e si sono scoperti gli instradamenti settatu dall'algoritmo di routing di PACMAN. Come si può vedere in Figura 2.2 i canali indesiderati, evidenziati in rosso sono il 50% del totale, senza considerare il canale proveniente dalla sorgente.

Sembra che il routing generi connessioni che, in realtà, non sono utili al funzionamento della rete; più nello specifico se esiste una proiezione fra una popolazione sorgente ed una obiettivo, vengono connessi tutti i neuroni dell'obiettivo con la sorgente; questo potrebbe essere pessimo a livello energetico e, quindi, si è pensato di

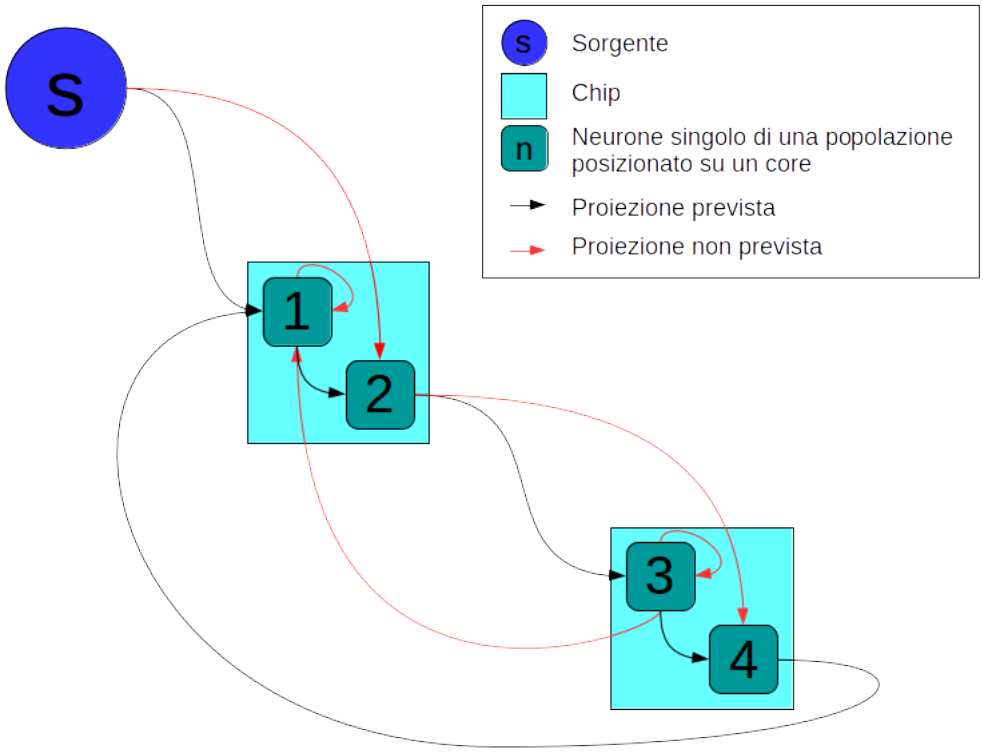

Figura 2.2: **Synfire Chain Test**: Schema della rete synfire posizionata su SpiNNaker e relative proiezioni ricollegate da PACMAN.

analizzare più a fondo il comportamento del routing con reti più grandi e valutarne l'impatto sui consumi. Quello che succede è che PACMAN, dopo aver diviso le popolazioni in sotto-popolazioni ha il compito di riconnetterle, ma, non avendo informazioni sulle sinapsi neurone-neurone, collega tutte le sotto-popolazioni della popolazione madre deferente a tutte le sotto-popolazione della popolazione madre afferente.

2.2 Definizione Algoritmo

L'idea è quella di confrontare le proiezioni generate da SpyNNaker derivate dalla descrizione della Cortical Microcircuit con gli instradamenti ottenuti dai router dopo che PACMAN ha partizionato e posizionato le popolazioni parziali sulla piattaforma neuromorfica. Da questo confronto si possono ottenere gli instradamenti errati generati dalla toolchain. L'energia dipende da quanti spike vengono emessi e gli spike dipendono dagli instradamenti perchè quando un neurone emette uno spike segue

sempre lo stesso percorso. Perciò possiamo ottenere gli spike errati di ogni neurone moltiplicando gli instradamenti errati relativi al medesimo neurone per il numero di spike emessi da esso. La seguente formula descrive il calcolo da effettuare:

$$E_{tot} = E_{R2C} \cdot n_{R2C} + E_{R2R} \cdot n_{R2R} \tag{2.1}$$

$$con \quad n_{R2*} = \sum_{pp_i}^{PP} \sum_{nj}^{N_i} \sum_{core_k}^{C} I_{n_j \to core_k} \cdot S_{n_j} \quad se \quad core_k \notin PR(n_j) \tag{2.2}$$

Dove pp_i è la popolazione parziale i-esime appartente all'insieme delle popolazioni complete PP, n_j è il neurone j-esimo dei neuroni inclusi in pp_i, $core_k$ è il processore k-esimo appartenente all'insieme di tutti i processori di SpiNNaker, $I_{n_j \to core_k}$ è l'instradamento tra n_j e $core_k$ e $PR(n_j)$ sono le proiezioni uscenti di n_j.

Nel lavoro proposto da Stromatias et al. [27] viene definito il consumo unitario medio per pacchetto, valore che ci servirà per avere una stima di energia consumata dai pacchetti.

Si è scelto di utlizzare, come strutture di memorizzazione, dei grafi ad albero. In questo modo si può tenere traccia degli instradamenti in maniera comoda, utilizzando i vertici per neuroni, router e processori e gli archi per gli instradamenti. In Figura 2.3 viene mostrato un esempio astratto di differenza fra grafo di una proiezione post-sinaptica e grafo di un instradamento su SpiNNaker; si supponga che il neurone sorgente sia lo stesso: la figura a sinistra mostra come SpyNNaker connette il neurone **N** alle popolazioni **1, 2, 3 e 4** e quella a destra rappresenta l'instradamento dallo stesso neurone (posizionato su un processore **M** ed appartenente ad una popolazione **Z**) alle popolazioni allocate sui processori (**a1**, **a2**, **b3**, **b3**) di SpiNNaker attraverso i router **Ra** e **Rb**.

C'è da sottolineare una questione cruciale prima di andare avanti nella discussione: la comunicazione fra 2 neuroni attraversa in ogni caso almeno un router; perciò i grafi d'instradamento sono formati sempre dal nodo sorgente che è sempre un neurone e da un router come secondo nodo; a valle del router il grafo può variare a seconda delle connessioni. Può collegare neuroni all'interno del processore (e di conseguenza del chip) di partenza, può collegare un altro router connesso a soli processori e può collegare vari altri router aumentando le diramazioni sui chip.

Tenendo a mente l'architettura di comunicazione di SpiNNaker appare chiaro che è importante differenziare gli instradamenti sotto l'aspetto energetico:

- Instradamenti R2R: si tratta dei collegamenti router-router che avvengono quindi fra chip distinti

- Instradamenti R2C: collegamenti router-core che avvengono all'interno di un singolo chip

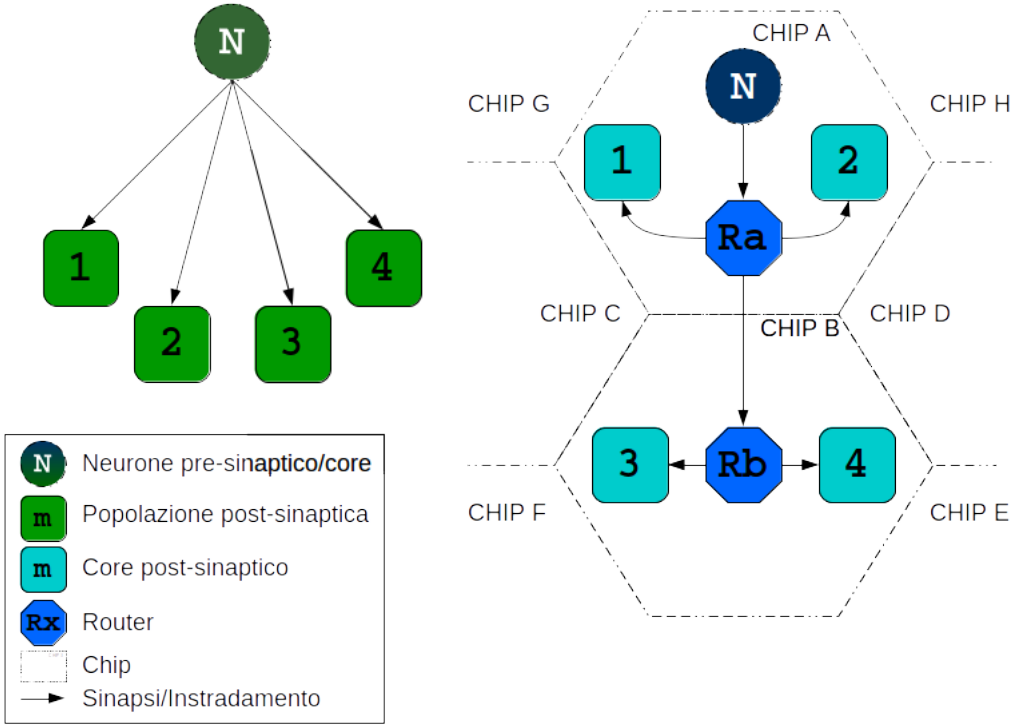

Figura 2.3: **Struttura dei Grafi**: A sinistra il grafo delle proiezioni in cui **N** è un neurone **1 2 3** e **4** sono popolazioni, a destra il grafo degli instradamenti dove: **Ca** e **Cb** sono chip, **Ra** ed **Rb** sono router, **1 2** sono popolazioni posizionate su processori di **a**, **3 4** sono popolazioni posizionate su processori di **b**.

Questa differenziazione è importante dato che l'impatto energetico è diverso tra i due instrademnti: uno di esso collega il router di un chip ad un router di un altro chip (comunicazione inter-chip) e l'altro collega il router ad un processore nello stesso chip (comunicazione intra-chip). Si presume, quindi, che il contributo energetico sia diverso. Non ci sono studi che mirano a investigare la differenza fra i contributi dei pacchetti R2C e R2R. Mi sono limitato, quindi, a calcolare i pacchetti lasciando i valori energetici come paramentro, così da poter in futuro usufruire di questo tool, differenziando i due contributi e avendo una stima più precisa dell'impatto energetico.

Per ottenere il numero di instradamenti R2C errati per ogni neurone si confrontano le foglie del grafo di instradamento con le foglie del grafo di proiezione che hanno come sorgente lo stesso neurone: se il grafo di instradamento ha più foglie di quello di proiezione si conta la differenza fra le foglie; questo è il numero di instradamenti R2C per neurone. Invece, per ricavare il numero di instradamenti R2R

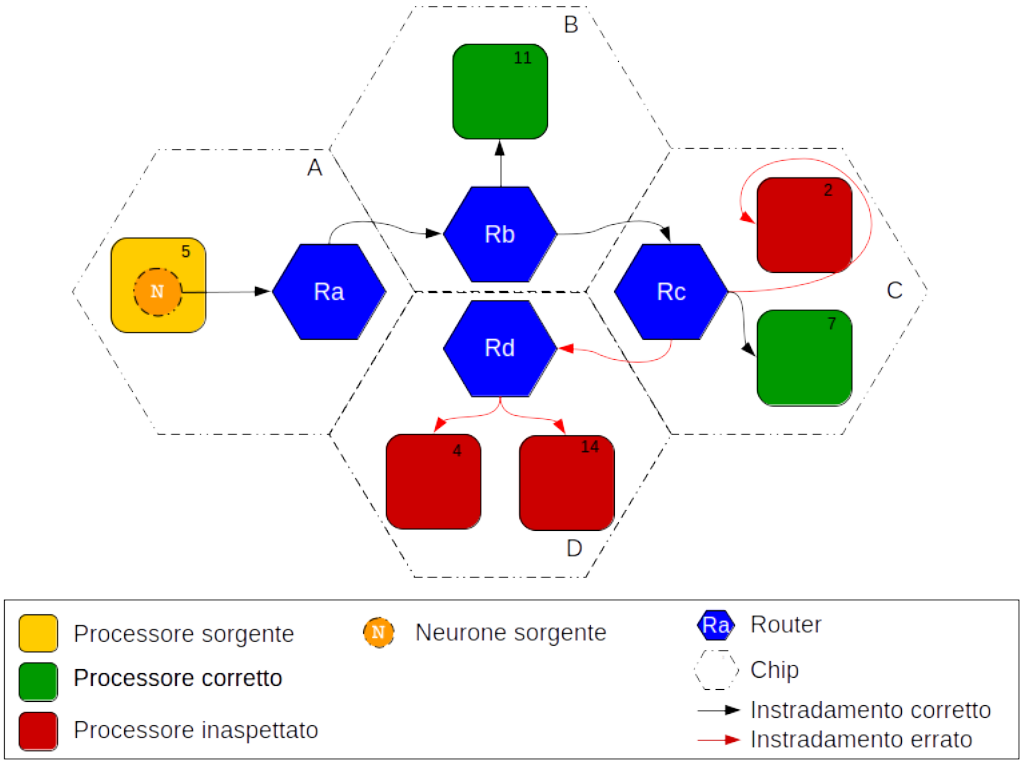

Figura 2.4: **Valid Off-Chip Route**: Grafo off-chip route valida anche se instrada verso un processore errato.

errati l'algoritmo meno banale si è agito in maniera diversa. Si supponga di avere una connessione R2R e che il secondo router colleghi un neurone con due processori: il primo processore è presente nel grafo di proiezione, il secondo no. In questo caso l'instradamento R2R è corretto; quindi, si considerano indesiderati solo i collegamenti R2R che hanno come archi finali solo collegamenti R2C indesiderati. La Figura 2.4 chiarisce eventuali dubbi rimanenti.

Quindi l'instradamento R2R viene conteggiato solo se tutti i core a cui arriva sono errati; si ottiene il numero totale di archi R2R errati percorrendo il grafo e contando tutti gli archi R2R che hanno a valle archi R2C errati.

Ottenuti il numero di instradamenti R2R e quello d'instradamenti R2C per neurone si possono moltiplicare questi valori per il numero di spike, ottenendo il numero di pacchetti R2R e R2C. Sommando tutti questi contributi per popolazione e moltiplicandoli per i corrispettivi valori energetici otteniamo il dispendio energetico aggiuntivo legato agli eventi sinaptici R2R e R2C per popolazione parziale.

In Figura 2.5 il flow chart per il calcolo dell'impatto energetico relativo agli eventi sinaptici indesiderati.

2.3 Creazione Framework

Questo algoritmo però ha bisogno di una struttura che gestisca tutte le variabili in gioco, data la complessità di SpiNNaker e delle reti che è possibile simulare con detta piattaforma. La struttura software di SpiNNaker permette di estrarre una quantità consistente di dati a fine simulazione: infatti, modificando dei flag nei file di configurazione prima di lanciare una simulazione è possibile ottenere in output file testuali che descrivono il posizionamento delle popolazioni parziali sui core, il consumo energetico nelle varie fasi della simulazione e gli indirizzi dei processori. Gli unici dati che non è possibile ottenere in output sono le routing table, ma è possibile riutilizzare la classe transceiver come già detto nella 2.1. Ottenuti questi dati è possibile analizzare il routing di SpiNNaker. Come spiegato nella sezione 1.5.2, i pacchetti multicast non contengono in sè la destinazione del pacchetto, ma solo l'id del neurone che invia il pacchetto. É il router che si occupa di confrontare questo id con le routing key che il router possiede e di inoltrare il pacchetto; per questo è necessario ricreare e automatizzare il processo di verifica dei pacchetti in ingresso al router per verificare se il neurone in analisi della popolazione sorgente è collegato effettivamente con la popolazione obiettivo che si sta analizzando. Per le strutture ad albero si sono utilizzati grafi direzionati presenti nel package NetworkX [4] e i metodi associati al package per le elaborazioni richieste che implementano visite e conteggi di archi, nonchè la possibilità di marcare archi e nodi con attributi. Si è costruito così un framework per gestire queste informazioni ed analizzare la quantità di instradamenti creati: la classe Python *EnergyModel* è stata creata per questo scopo. Dato che l'elaborazione delle informazioni partono da file di report generati dalla toolchain di SpiNNaker è necessario che questa classe venga richiamata dopo l'esecuzione della simulazione, ma prima della fine della simulazione perchè allo stesso tempo comunica con SpiNNaker attraverso la classe transceiver per ottenere le tabelle di instradamenti direttamente dai router dei chip, che una volta terminata la simulazione, smettono di funzionare.

Come si vede in Figura **??** il metodo *init* carica le informazioni dai file di report di SpiNNaker utilizzando i metodi *load_energy_variables* che carica i valori energetici relativi ai pacchetti multicast corretti, il tempo di simulazione effettivo e l'energia totale spesa dalla board, *load_placement_by_vertex* che genera un dizionario che a seconda della popolazione parziale inserita restiruisce la coppia chip, processore di appartenenza e *load_key_space* ed estrae tutte le tabelle di routing chiamando il metodo *create_routing_dictionary* realizzando un dizionario che ha come chiavi i chip e come valori le tabelle di routing corrispondenti. Ognuno di questi metodi ritorna ad *init* le informazioni come variabili di classe, così da averle a disposizione nella seguente elaborazione. Quando viene dichiarata un'instanza di *EnergyModel*, quindi, il metodo *init* richiede la versione del software di PyNN, la versione di SpiNNaker e il suo indirizzo url; questo permette il calcolo su

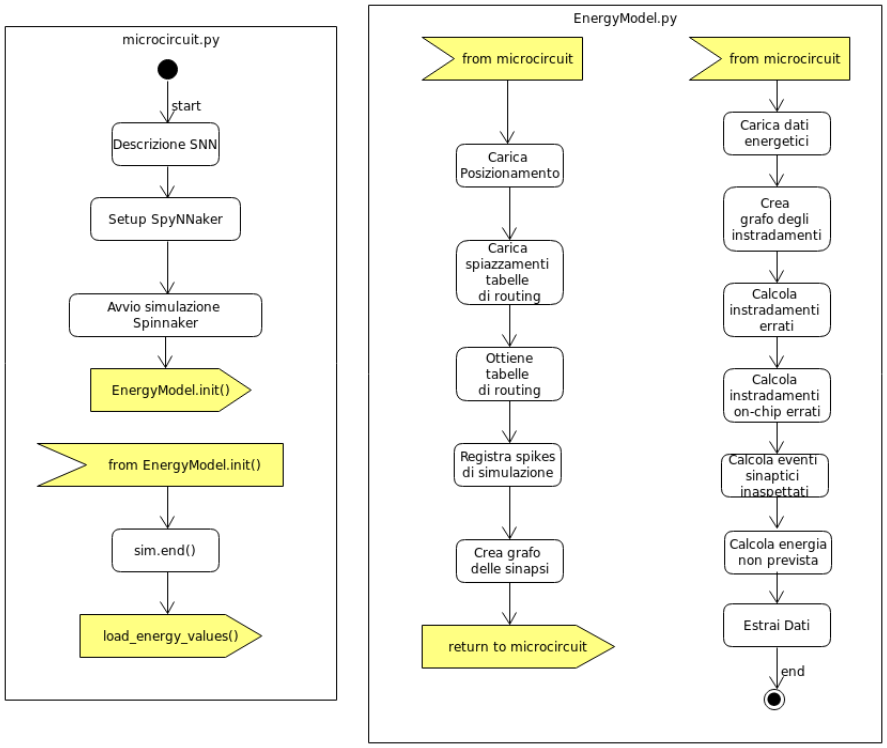

Figura 2.5: **Diagramma di flusso per impatto energetico del framework**.

ee

configurazioni diverse di SpiNNaker e di richiedere le tabelle di routing tramite il transceiver. Sono stati implementati due metodi pubblici nella classe EnergyModel: *create_routes_graphs_dict* e *find_unexpected_synaptic_events*. Il primo richiede la lista delle popolazione dichiarata nella rete con PyNN; questo metodo utilizza i sottometodi delle istanze di popolazione PyNN per ottenere le liste di spikes di ogni popolazione che vengono salvate in un dizionario globale di classe. Il singolo conteggio di spike per neurone viene poi salvato e assegnato al grafo di instradamento appartenente ad esso per essere poi utilizzato nel calcolo degli eventi sinaptici. Inoltre è proprio questo metodo a creare i grafi d'instradamento chiamando il sottometodo *create_routes_graph*: esso prende in ingresso il nome di una popolazione e per ogni popolazione parziale associata a quella totale trova le coordinate del processore su SpiNNaker utilizzando il dizionario di posizionamento creato da *init*. A questo punto scorre tutti i neuroni presenti nella popolazione parziale e ne crea il primo nodo-neurone, il router relativo e l'arco tra essi e calcola la source key sommando l'id number del neurone e l'indirizzo relativo alla popolazione parziale. Poi viene richiamato ancora un sotto metodo che estrae le routing table dal dizionario di routing globale, confronta detta source key con tutte le routing key di

ogni routing table e nel caso in cui ci sia match con una routing key collega al router del neurone sorgente tutti i processori e tutti i router eventualmente dichiarati nella routing key; nel caso ci siano router dichiarati nella routing key verrano linkati anche i processori connessi a questi router. Al termine della creazione del grafo, esso viene aggiunto ad un dizionario globale per essere utilizzato nell'elaborazione successiva. Il secondo metodo è quello che effettivamente estrapola i dati utili per l'analisi. Esso ha come variabile d'ingresso la lista di proiezioni sinaptiche delle popolazioni dichiarate nella fase d'istanziazione della rete. Una piccola parentesi è d'obbligo prima di continuare con la descrizione del framework. Da ogni proiezione dichiarata con PyNN si possono estrarre solo le popolazioni che la proiezione collega, ma le informazioni utili all'elaborazione sono le connessioni neurone-neurone per costruire il grafo delle sinapsi. La soluzione adottata è stata quella di modificare la toolchain di sPyNNaker per la creazione delle proiezioni, istanziando un dizionario globale come variabile privata che salva le connesioni tra neuroni nella classe "Projections". Diventa banale, quindi, estrarre le coppie di neuroni collegate dalle proiezioni. A questo punto avviene il vero confronto descritto nell'algoritmo: NetworkX possiede dei metodi che rendono semplice la visita del grafo e il conteggio degli archi. Si conteggiano gli archi come descritto nella sezione 2.2. Vengono contati anche gli archi per numero di hop: questo ci dà un'informazione sui salti che compiono gli instradamenti derivati dal chip preso in esame. In questa fase viene calcolato anche il contributo energetico, moltiplicando gli instradamenti indesiderati per il numero di spike e per il valore di energia per pacchetto, rispettivamente R2R e R2C.

Si estraggono, quindi:

- il numero di pacchetti indesiderati R2R e R2C per popolazioni parziali
- il numero di pacchetti desiderati R2R e R2C per popolazioni parziali
- un lista instradamenti R2R ordinati per numero di hops divisi per chip
- l'energia spesa dai pacchetti indesiderati R2R ed R2C divise per popolazioni parziali

2.4 Risultati

In questa sezione vengono riportati i grafici a barre delle simulazioni effettuate con reti Cortical Microcircuit al 5%, 10%, 15% e 20% con 75 neuroni per processore. I diagrammi a barre a stack come quello in Figura 2.6 hanno sulle ascisse gli id dei chip e sull ordinate la percentuale di hop su totale dei canali uscenti dal chip. Il colore delle barre rappresenta la *profondità* di questi canali. La linea rossa rappresenta il numero assoluto di canali per chip.

I diagrammi a barre con barre doppie rappresentano l'energia totale spesa per chip per inviare i pacchetti indesiderati, rispettivamente R2C ed R2R. 2.6

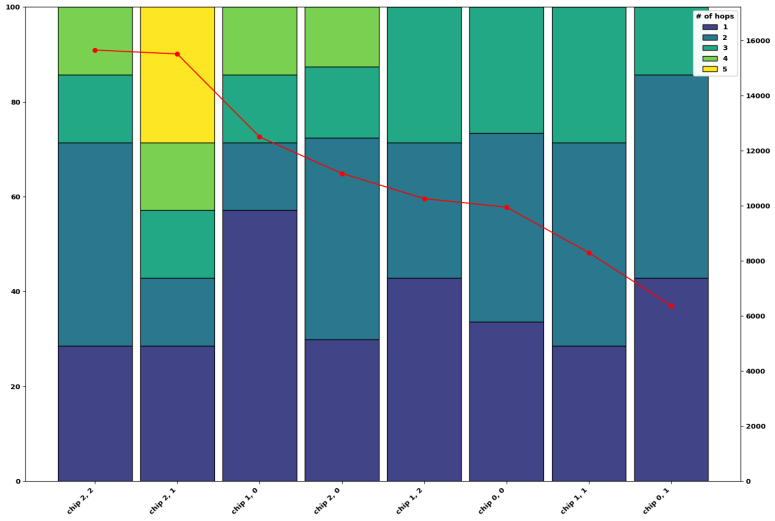

Figura 2.6: **Diagramma a barre che riporta numero di hops per chip ed elongazione sinaptica di una rete Cortical Microcircuit al 5%**:.

Ho ordinato entrambi i grafici in modo che riportassero lo stesso ordine dei chip in ascissa, così da permettere un confronto.

Si vede da 2.6 che un CM5% viene posizionata su 8 chip e almeno il 15% degli instradamenti raggiungono 3 hop (chip 0,1) fino ad arrivare ad un 25% di canali che raggiungono 5 hop dal chip 2,1.

Come si vede da 2.7 una maggiore lunghezza sinaptica non corrisponde sempre ad un maggiore dispendio di energia. Questo perchè alcune popolazioni hanno un fire rate più elevato e questo fa crescere l'energia consumata. Notiamo anche che c'è una relazione costante fra energia R2C e R2R: questo ci dice che in questa rete il rapporto fra spike inter-chip e intra-chip è stabile.

Si vede da 2.8 che un CM10% viene posizionata su 16 chip e almeno il 15% degli instradamenti raggiungono 4 hop (chip 2,2) fino ad arrivare ad un 20% di canali che raggiungono 8 hop dal chip 0,3.

Dalla Figura 2.9 è ben visibile che l'apporto energetico è basso, anche se da 2.8 vediamo che è al secondo posto per elongazione sinaptica. É chiaro che la quantità di spike emesso ha un peso notevole per ricavare l'energia.

Con la simulazione al 20% SpiNNaker alloca 30 chip e raggiungere un hop massimo di 8.

Nella simulazione CM20% i chip che impattano maggiormente sono quasi tutti quelli che hanno maggior elongazione sinaptica, si può dedurre che da una certa soglia di quantità di part population il contributo di questo fattore comincia a

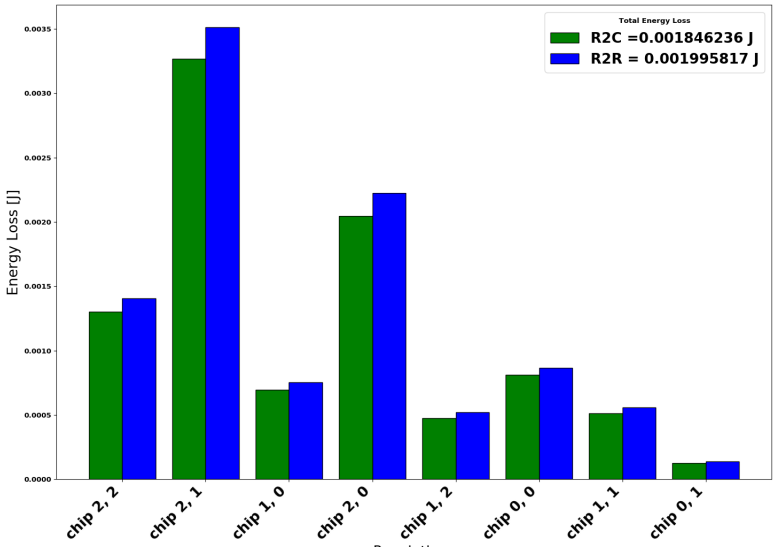

Figura 2.7: **Diagramma a barre che riporta la quantità di energia spesa dai pacchetti R2C e R2R per chip di una rete Cortical Microcircuit al 5%.**

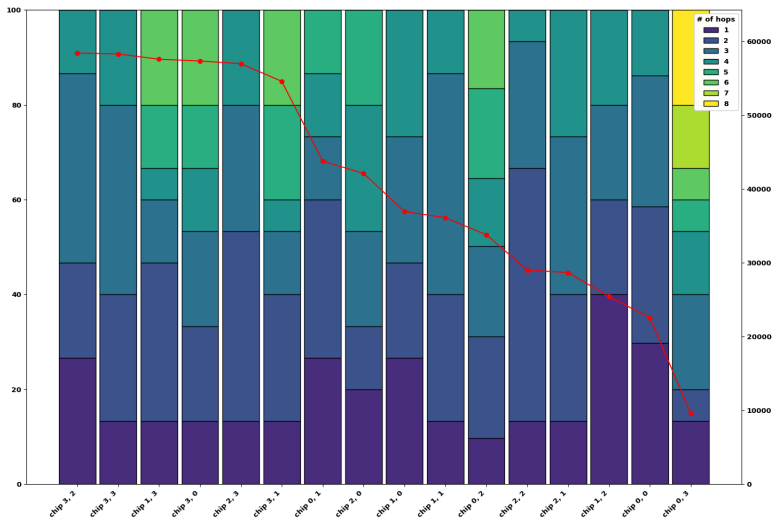

Figura 2.8: **Diagramma a barre numero di hops per chip ed elongazione sinaptica si una rete Cortical Microcircuit al 10%.**

impattare maggiormente sull'energia consumata.

Notare che in tutte le simulazioni non c'è relazione fra elongazione sinaptica e

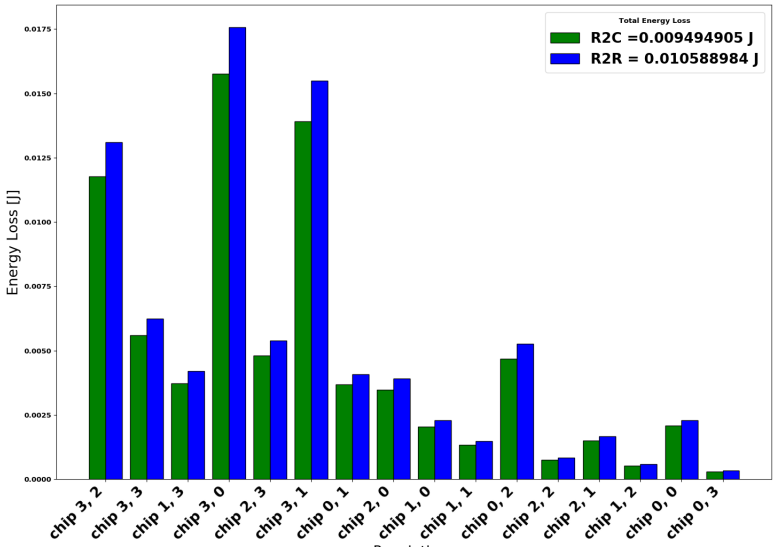

Figura 2.9: **Diagramma a barre che riporta la quantità di energia spesa dai pacchetti R2C e R2R per chip di una rete Cortical Microcircuital 10%.**

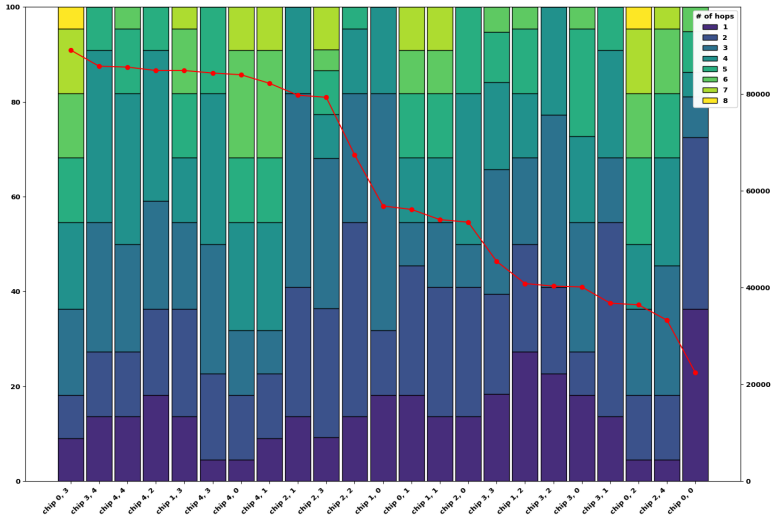

Figura 2.10: **Diagramma a barre numero di hops per chip ed elongazione sinaptica di una rete Cortical Microcircuit al 15%.**

canali negli hop più vicini al chip: questo è sintomo di un partizionamento fatto senza tenere conto della connettività.

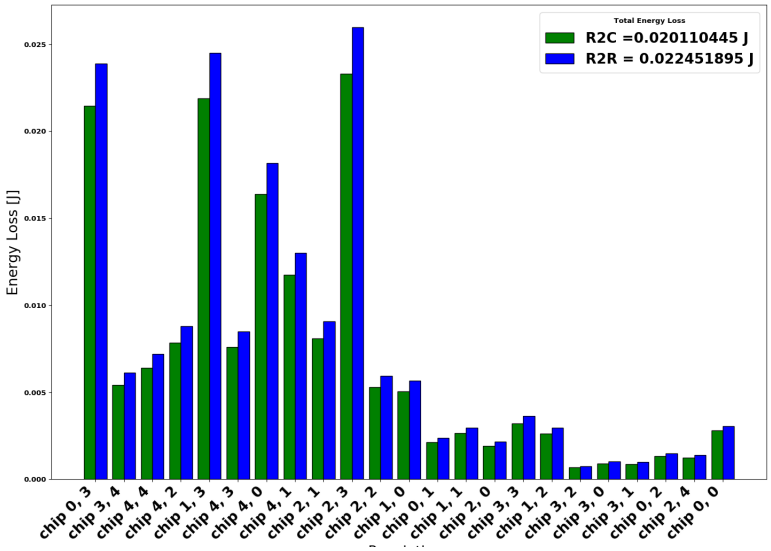

Figura 2.11: **Diagramma a barre che riporta la quantità di energia spesa dai pacchetti R2C e R2R per chip di una rete Cortical Microcircuital al 15%.**

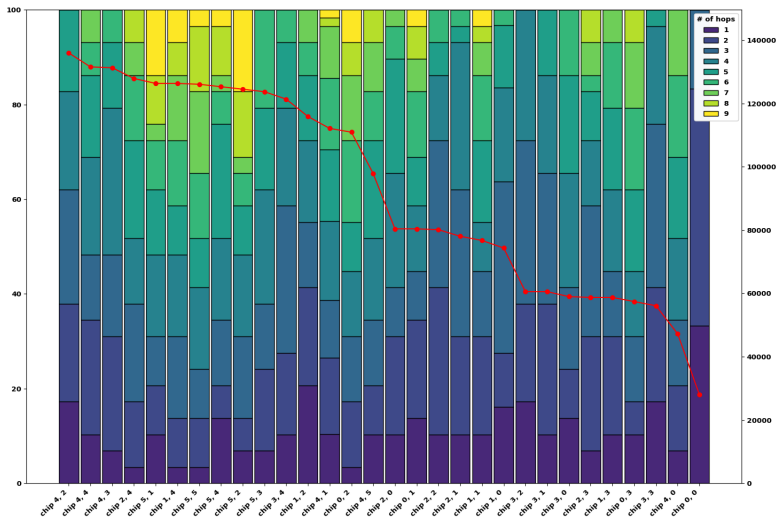

Figura 2.12: **Diagramma a barre numero di hops per chip ed elongazione sinaptica di una rete Cortical Microcircuit al 20%.**

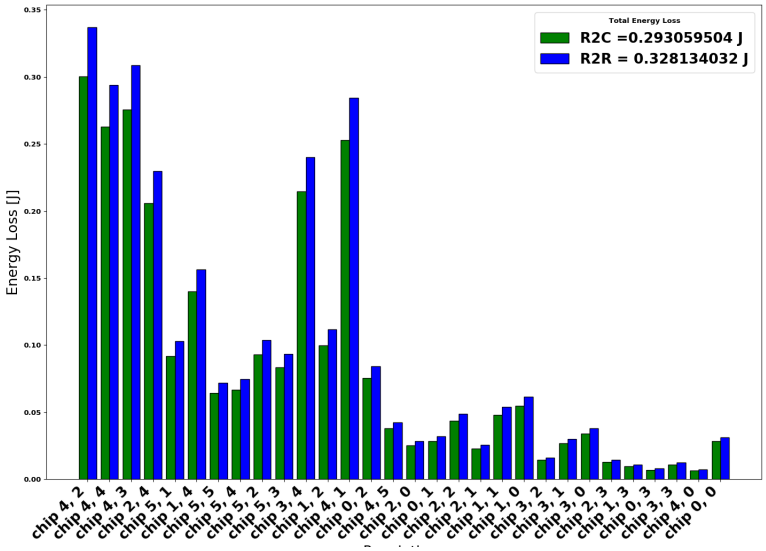

Figura 2.13: **Diagramma a barre che riporta la quantità di energia spesa dai pacchetti R2C e R2R per chip di una rete Cortical Microcircuit 20%.**

Prendendo come esempio la rete al 15% ho trovato un numero di pacchetti indesiderati pari a 20084. Il report energetico dà come quantità di pacchetti multicast inviati un numero pari a 52251; ciò sta a significare che, assumendo 8 nJ il consumo energetico per pacchetto [20] abbiamo un consumo atteso nel tempo di simulazione di 418 μJ ed un consumo inatteso 161 μJ, per cui l'impatto energetico dei pacchetti multicast inattesi raggiunge il 40,75%, per un CM al 15%. Di questo consumo un 47,25% va attribuito ai pacchetti R2C ed il restante 52,75% ai pacchetti R2R.

Capitolo 3

Analisi impatto energetico del posizionamento

Come si può vedere da [28] applicando tecniche di partizionamento come lo Spectral Clustering si può ridurre fino al 15% il traffico tra i nodi di SpiNNaker rispetto al posizionamento effettuato dalla tool-chain integrata PACMAN. Da questo risultato ho voluto indagare se ci fosse ancora margine di miglioramento su detto traffico. Perciò ho provato a fare un confronto teorico tra gli algoritmi di posizionamento più promettenti allo stato dell'arte per avere un'idea di quanto si può ancora migliorare l'efficienza di comunicazione dei pacchetti su SpiNNaker con l'obiettivo di integrare il più performante nella tool-chain ufficiale.

3.1 Metodo

Utilizzando PyNN una SNN può essere rappresentata come un grafo di Popolazioni e Proiezioni, mentre l'architettura neuromorfica può essere rappresentata come un grafo pesato e non diretto i cui nodi sono i processori e gli archi sono le connessioni fra di loro. Si è deciso di inserire un vincolo per il quale solo popolazioni parziali dello stesso tipo possono essere posizionate sullo stesso chip. Ci sono però due vincoli che, invece, dipendono dall'architettura di SpiNNaker:

1. un processore può simulare solo neuroni appartenenti alla stessa popolazione (*vincolo di appartenenza*)

2. il numero di neuroni simulabili da ogni processore è limitato e dipende dalla complessità del modello (*vincolo di limite*).

La metrica da minimizzare è chiamata lunghezza sinaptica globale e descrive l'inefficienza della comunicazione dovuta alla distanza tra due entità che comunicano dopo essere state posizionate. Possiamo ridefinire il problema per soddisfare i 2 vincoli. Il

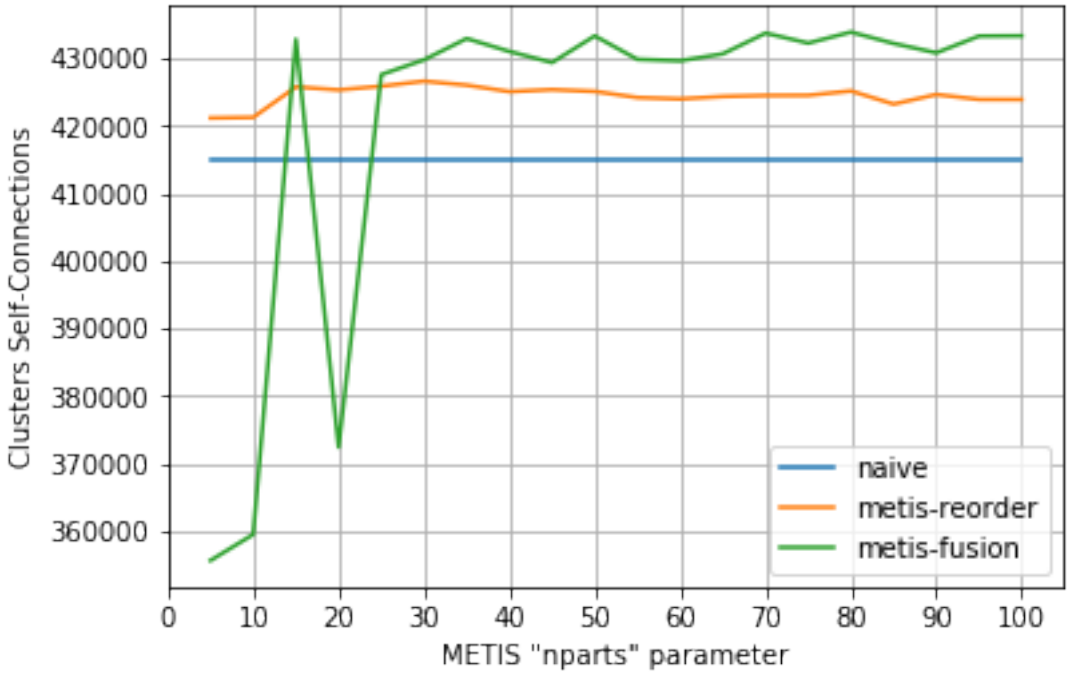

Figura 3.1: **Esplorazione del parametro *nparts* con CM10%**: la tecnica metis-fusion si comporta meglio delle altre raggiunta una certa soglia di grandezza del cluster.

primo passo è ottenere un grafo delle popolazioni parziali: si divide ogni popolazione originaria cercando di massimizzare la connettività delle sub-popolazioni. Il secondo passo prevede di assegnare ogni popolazione parziale ad un chip, sovrapponendo in modo efficiente il grafo delle popolazioni parziale con quello dei chip.

3.1.1 Partizionamento

Per il partizionamento delle popolazioni ho usato il tool di Metis che applica il multilevel k-way partitioning. Uno dei parametri interessanti di Metis è l'iperparametro *nparts*: esso definisce quanto devono essere grandi clusters restituiti dal tool. Ho ottenuto per quali valori di questo parametro si massimizzava la self-connectivity di una rete di test; questo metodo minimizza il taglio del grafo delle popolazioni 3.1.

Rilassando il vincolo di appartenenza, si può applicare Metis sul grafo dei neuroni. In uscita otteniamo i neuroni etichettati con la popolazione e il cluster Metis di appartenenza. I cluster restituiti da Metis non sono posizionabili su SpiNNaker perchè sono cluster in cui sono presenti neuroni con caratteristiche diverse e, quindi, sarebbe violato il primo vincolo. Perciò serve una tecnica che "legalizzi" questo

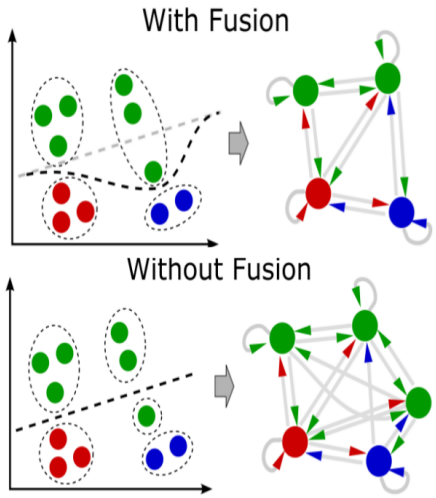

Figura 3.2: **Passi per la tecnica "Fusion"**: Questa procedura permettere di accorpare cluster incompleti per ottimizzare l'uso delle risorse.

partizionamento. Ho sviluppato un riordinamento in grado di creare popolazioni parziali posizionabili su SpiNNaker. Si cerca di aggregare i neuroni rispettando prima di tutto il vincolo di appartenenza e poi cercando di rispettare i cluster Metis. Per fare ciò si è riordinata la matrice di adiacenza del grafo dei neuroni. Dopo il riordinamento si ottengono le popolazioni parziali, ma si perde parzialmente l'informazione sul clustering. Inoltre, la generazione di popolazioni parziali il cui numero è molto minore al limite massimo di neuroni per chip porta ad un peggioramento dell'efficienza del placement: l'uso non ottimizzato delle risorse condurrebbe ad un aumento del traffico di pacchetti. La tecnica *fusion* cerca di rimediare a questi problemi: selezione due cluster intra-popolazione e li unisce se la somma dei neuroni di questi cluster è simulabile su un processore. Questo viene fatto però partendo dal cluster più piccolo in modo crescente per evitare piccoli nodi con grande connettività che degradano molto le prestazioni. In Figura 3.2 è mostrato un semplice esempio, dove un neurone appartenente alla popolazione verde è clusterizzato con neuroni della popolazione blu. La procedura di fusione riconosce il neurone verde e lo riassegna ad uno dei sotto cluster, evitando la creazione di nodi supplementare nel grafo delle popolazioni parziali.

3.1.2 Posizionamento

Una volta ottenuto il partizionamento è necessario posizionare il grafo delle popolazioni parziali sul grafo dei chip. Abbiamo sviluppato tre strategie di partizionamento:

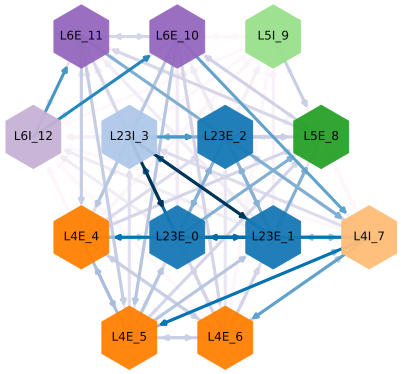

Figura 3.3: **Rappresentazione del placement naive**: le popolazioni di colore uguale fanno parte dello stesso layer; quelle chiare sono inibitore, quelle scure eccitatorie. Le frecce indicano, invece, la quantità di connessioni fra le popolazioni: più è scura la freccia più le popolazioni sono connesse.

- Spectral Embedding: attua un'analisi spettrale del grafo ed applica una riduzione della dimensione ottenendo una sua rappresentazione planare
- Scotch: usa i programmi disponibili nell'omonima software suite [22] per generare il posizionamento attraverso un Dual-Recursuve Bi-partitioning.
- Simulated Annealing: usato per trovare l'ottimo di problema di ottimizzazione [17].

Si è implementato anche un posizionamento che simula il comportamento di PACMAN: il grafo dei processori è stato ordinato seguendo un sistema di coordinate polari (ρ, φ) partendo da un chip a scelta. Il raggio $\rho = max(|x|, |y|, |x - y|)$ è stato calcolato usando la distanza esagonale. L'angolo $\varphi \in [0, 2\pi)$ è espresso in radianti. La procedura inizia a posizionare le popolazioni parziale nel chip di partenze e poi cambia chip quando tutti i processori dentro un chip sono pieni. Al crescere di ρ, le sub-popolazioni vengono distribuiti a spirale e saranno separati da una distanza crescente. Questo posizionamento ci servirà per fare un confronto con le nuove tecniche.

La procedura che usa lo Spectral Embedding attua un'analisi spettrale del grafo ed una riduzione dimensionale per ottenere una sua rappresentazione planare. In

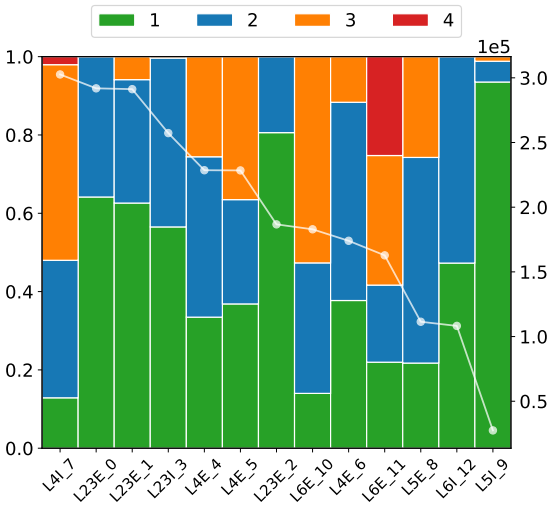

Figura 3.4: **Grafico a barre per il placement naive**: A sinistra la percentuale di connessioni sul totale di connessioni per chip. A destra il valore di elongazione per chip.

questo modo il grafo delle popolazioni così ottenuto può essere direttamente sovrapposto col grafo dei chip. L'associazione tra le popolazioni parziali e i chip è ottenuta attraverso la risoluzione di un Integer Linear Programming (ILP). La procedura inizia estraendo i primi 5 autovalori, ed i relativi autovettori dalla matrice laplaciana $\mathbf{L}$. Gli autovalori formano una matrice Λ che rappresentaza le popolazioni parziali in uno spazio R^5. La riduzione non-lineare si ottiene usando *Sammon Mapping* ottenendo uno spazio in R^2 L'algoritmo *Sammon Mapping* minimizza la funzione d'errore E dove d_{ij} è la distanza nell'autospazio R^5 e d^*_{ij} è la distanza nello spazio bidimensionale [26].

$$E = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij} - d^*_{ij})^2}{d_{ij}} \tag{3.1}$$

Ogni chip nella mesh è rappresentato da un punto x, y in un sistema a coordinate assiali. Si sovrappone quindi il grafo G_T sul grafo G_{pp} proiettando la mesh dei chip nello spazio del posizionamento utilizzando la formula seguente:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \sqrt{\frac{2A_h}{3\sqrt{3}}} \begin{pmatrix} \sqrt{3} & -\frac{\sqrt{3}}{2} \\ 0 & \frac{3}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.2}$$

Dove x, y è la coordinata del chip nella mesh esagonale, x^*, y^* è la coordinata nello spazio del posizionamento. La lunghezza del lato dell'esagono è usata come fattore di normalizzazione e calcolato isando l'area dell'esagono $A_h = \frac{A}{m}$ occupata

da ogni chip. Il fattore di normalizzazione permette di scalare la mesh dei chip rispetto all'area A occupata dalle popolazioni parziali. Dopo aver proiettato i punti nello spazio del posizionamento, vengono traslati per essere centrati sulla mediana dei punti che rappresentano le popolazioni parziali. Adesso possiamo descrivere il problema del posizionamento usando la formulazione ILP.

$$minimizzare \quad f(X) \qquad f : \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} d_{i,j} \tag{3.3}$$

$$dove \qquad \sum_{i=1}^{n} x_{i,j} <= k \quad \forall j \in 1, ..., m \tag{3.4}$$

$$\sum_{j=1}^{m} x_{i,j} \quad \forall i \in 1, ..., n \tag{3.5}$$

Dove la matrice $\mathbf{X} = (x_{i,j})$, con $x_{i,j} \in \{1,0\}$ è la matrice del piazzamento. Se $x_{i,j} = 1$, la popolazione parziale i è mappata sul processore j. Ora abbiamo due problemi:

- Ogni processore target può ospitare al massimo k popolazioni parziali
- Ogni popolazione parziale può essere associata soltanto ad un processore target

Il problema ILP è stato modellato usando la libreria Python library PuLP e risolto con il risolutore *COIN-OR branch and cut*. Scotch è una procedura di mapping che usa l'omonima suite software (SCOTCH). Il Dual Recursive Bipartitioning (DRB) è la procedura usata da questo tool [23]. The DRB può usare una quantità elevata di altri metodi di bipartizionamento secondo la strategia definita dall'utente o dedotte dalle propietà del grafo. I principali metodi disponibili sono: Gibbs-Poole-Stockmeyer [11], Fiduccia-Mattheyses [8], Greedy Graph Growing [16] e Diffusion [21].

Diminuisce quindi la connettività fra le popolazione più lontane, segno che questa tecnica opera nella direzione giusta diminuire i pacchetti scambiati.

Il flusso di posizionamento di SCOTCH pre-partiziona il grafo dei chip attraverso il programma `amk_grf`. Il programma `amk_grf` prende in innput un grafo in formato *grf* e crea un file target (formato *tgt*) che contiene un'architettura target scomposta della stessa topologia del grafo in ingresso. Una volta ottunuta la scomposizione del grafo target, il grafo delle popolazioni parziali è posizionato usando il programma *gmap*. Il programma *gmap* prende in ingresso il grafo delle popolazioni parziali in formato *grf* e il grafo dei chip in formato *tgt* ed esedue la procedura DRB minimizzando la funzione di costo della comunicazione, simile all'elongazione sinaptica globale. Il file d'uscita di *gmap* (*map* format) che contiene l'associazione tra i nodi sorgente ed i nodi target. Ho sviluppato un modulo Python in grado di esportare

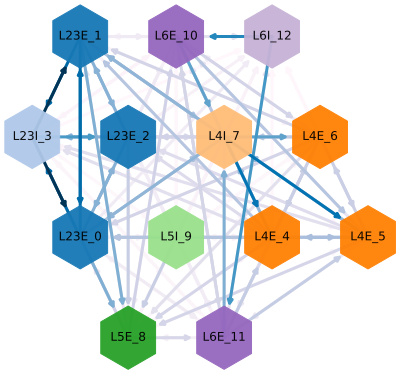

Figura 3.5: **Rappresentazione del placement con Spectral Embedding**: le popolazioni di colore uguale fanno parte dello stesso layer; quelle chiare sono inibitore, quelle scure eccitatorie. Le frecce indicano, invece, la quantità di connessioni fra le popolazioni: più è scura la freccia più le popolazioni sono connesse.

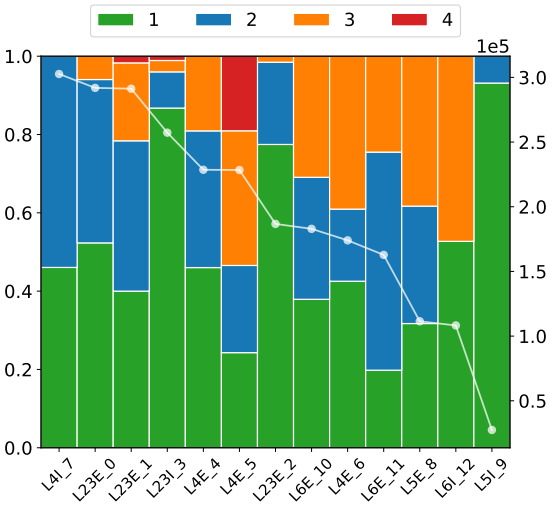

Figura 3.6: **Grafico a barre per il placement con Spectral Embedding**: A sinistra la percentuale di connessioni sul totale di connessioni per chip. A destra il valore di elongazione per chip.

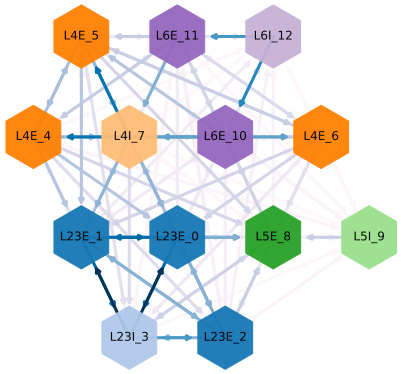

Figura 3.7: **Rappresentazione del placement con SCOTCH**: le popolazioni di colore uguale fanno parte dello stesso layer; quelle chiare sono inibitore, quelle scure eccitatorie. Le frecce indicano, invece, la quantità di connessioni fra le popolazioni: più è scura la freccia più le popolazioni sono connesse.

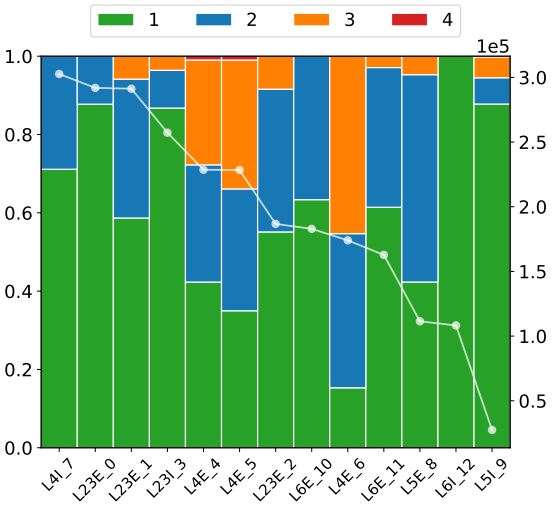

Figura 3.8: **Grafico a barre per il placement con SCOTCH**: A sinistra la percentuale di connessioni sul totale di connessioni per chip. A destra il valore di elongazione per chip.

un grafo NetworkX su un file secondo il formato *grf* usato da SCOTCH e capace di automatizzare la procedura appena descritta.

Simulated Annealing (SA) è una tecnica ben conosciuta per trovare soluzioni di un problema di ottimizzazione [17]. Per utilizzare SA è conveniente esprimere l'allungamento sinaptico globale in forma matriciale e definire una funzione di costo da minimizzare. Dato il grafo delle popolazioni parziali G_{pp}, costruiamo la sua matrice di adiacenza $\mathbf{A} = (a_{ij})$ descritta come segue:

$$a_{ij} = \begin{cases} w_{ij} & se \exists (v_i, v_j) \in E_{pp} \\ 0 & altrimenti \end{cases} \qquad \forall i, j \in 1, ..., n \tag{3.6}$$

Dato il grafo dei processori G_T costruisco la sua matrice delle distanze $\mathbf{D} = (d_{ij})$ dove ogni d_{ij} è la lunghezza del percorso minimo fra due processori cpu_i e cpu_j. La matrice delle distanze può essere costruita con gli argoritmi di Floyd-Marshall o ripetendo l'algoritmo di Dijkstra's se $|E_T| << |V_T|^2$ assumendo di avere tante subpopolazioni quanti sono i processori e una regoladi routing $\Pi : v_1, ..., v_n \rightarrow cpu_1, ..., cpu_n$ costruisco il *vettore di permutazione* $\pi : (\Pi(v_1), ..., \Pi(v_n))$ e la *matrice di permutazione* $\mathbf{P}_\pi = (p_{ij})$:

$$p_{ij} = \begin{cases} 1 & se \quad i = \pi_j \\ 0 & altrimenti \end{cases} \qquad \forall i, j \in \{1, ..., n\} \tag{3.7}$$

La matrice di permutazione è applicata a $\mathbf{D}$ per permutare le sue righe e le sue colonne. Ottengo la matrice $\mathbf{D}_\pi = \mathbf{P}_\pi \mathbf{D} \mathbf{P}_\pi$. L'elonganzione sinaptica globale può essere espressa in forma matriciale e usata come funzione di costo per l'algoritmo SA:

$$f : \mathbf{e^T}(\mathbf{A} \odot \mathbf{D}_\pi)\mathbf{e^T} = \sum_{i,j} a_{ij} * d_{ij}^{(\pi)} \tag{3.8}$$

Dove $\odot$ è la moltiplicazione elemento per elemento ed $\mathbf{e}$ è un vettore colonna i cui cui elementi sono tutti uguali a 1. Ho usato l'implementazion SA fornita nell'ecosistema SciPy usando il parametro *temperature* per decidere quanti elementi del vettore di permutazione π scambiare.

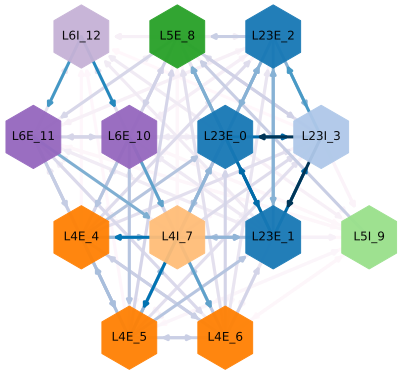

Figura 3.9: **Rappresentazione del placement con Simulated Annealing**: le popolazioni di colore uguale fanno parte dello stesso layer; quelle chiare sono inibitore, quelle scure eccitatorie. Le frecce indicano, invece, la quantità di connessioni fra le popolazioni: più è scura la freccia più le popolazioni sono connesse.

3.2 Risultati

Di seguito riportiamo le tabelle dei risultati per una rete CM al 10%:

Placement	Partitioning		Impatto
	Naive	Fusion	
Naive	+8%	+12%	3.23%
Spectral Embedding	+14%	+21%	8.62%
SCOTCH	+17%	+22%	5.83%
Simulated Annealing	+23%	+29%	7.37%

Come vediamo in 3.10 il posizionamento tramite Spectral Annealing in combinazione con la tecnica MetisFusion è la più efficiente tra le tecniche studiate in questa trattazione. É molto interessante vedere come in Figura 3.10 il 40% dei canali su ogni chip è racchiusa entro il primo hop per il 90% delle popolazioni e che il 70% dei canali su ogni chip è racchiusa nei primi 2 hop tranne che per la popolazione L5I_9 che è l'unica ad avere canali con 4 hop.

Otteniamo, quindi, l'elongazione sinaptica globale della rete per ogni metodo di posizionamento; questa è direttamente proporzionale all'energia spesa derivante dai

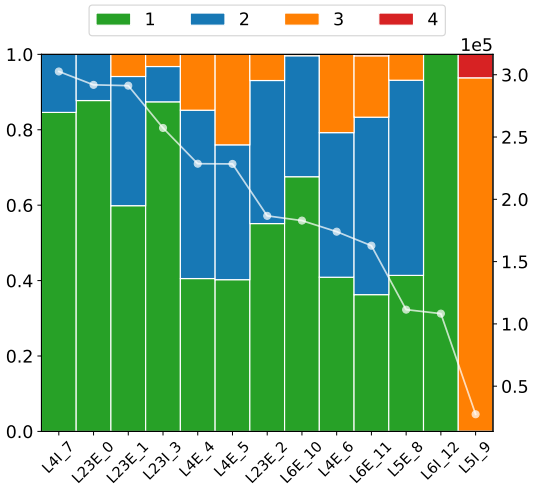

Figura 3.10: **Grafico a barre per il placement con Simulated Annealing**: A sinistra la percentuale di connessioni sul totale di connessioni per chip. A destra il valore di elongazione per chip.

pacchetti R2R. Perciò per avere una stima di energia spesa da ogni metodo sulla stessa rete basta moltiplicare questa metrica per il consumo unitario del pacchetto. Assumendo il consumo energetico unitario per pacchetto di 8 nJ otteniamo un consumo di 36,95 J per una rete CM al 10%.

Capitolo 4

Conclusioni

Il framework sviluppato in questa tesi ha il compito di analizzare l'instradamento, il partizionamento ed il posizionamento di SNN su SpiNNaker, valutando la presenza di pacchetti indesiderati generati dall'architettura. La creazione da parte di questo framework degli alberi di propagazione degli spike permette, infatti, di capire se questi spike raggiungono processori a cui non sono diretti. Questo è stato possibile costruendo un modello che simuli il comportamento di un Router SpiNNaker e la relativa generazione degli instradamenti. Estraendo le routing key dalla simulazione di una SNN si è potuto testare il simulatore di Router, validandone il funzionamento. A questo punto si sono costruiti gli alberi di propagazione degli spike e valutata questa propagazione, cercando eventuali rami dell'albero che conducessero a destinazioni indesiderate. Effettivamente, si è notato che queste destinazioni esistono sempre e che portano alla generazione di pacchetti indesiderati che si propagano su tutta l'architettura. L'impatto dei pacchetti indesiderati arriva fino al 40% dei pacchetti totali inviati. Questo impatta anche sull'energia spesa per l'invio di questi pacchetti, calcolabile sapendo il consumo energetico unitario del pacchetto. Questo comportamento è dovuto agli algortmi di partizionamento, posizionamento e instradamento presenti nella tool-chain SpiNNaker. Per eliminare questo surplus di pacchetti ssi sono considerate tecniche alternative a questi algoritmi. Si sono selezionati promettenti algoritmi di partizionamento e posizionamento e si è creato un modello per testarne l'efficienza: teoricamente questi permettono di ridurre lo scambio di pacchetti del 28Il tool sviluppato è stato pensato per lavorare su una scheda singola (Spin5), ma può facilmente essere adattato per lavorare su architetture SpiNNaker-based più grandi. In futuro ci si propone di implementare queste tecniche alternative di partizionamento e posizionamento nella tool-chain dell'architettura e valutare l'impatto effettivo sulle comunicazioni tra i processori.

Ringraziamenti

Grazie

Bibliografia

[1] Nicolas Brunel and Mark CW Van Rossum. Lapicque's 1907 paper: from frogs to integrate-and-fire. *Biological cybernetics*, 97(5-6):337–339, 2007.

[2] Jing-Chao Chen. Dijkstra's shortest path algorithm. *Journal of Formalized Mathematics*, 15(9):237–247, 2003.

[3] Andrew Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: A common interface for neuronal network simulators. 2:11, 02 2008.

[4] NetworkX developers. Networkx, 2014-2018.

[5] Mikael" "Djurfeldt. "the connection-set algebra: a formalism for the representation of connectivity structure in neuronal network models, implementations in python and c++, and their use in simulators". *"BMC Neuroscience"*, "12"("1"):"P80", "Jul" "2011".

[6] Matthias Ehrlich, Christian Mayr, Holger Eisenreich, Stephan Henker, Andre Srowig, Andreas Grübl, Johannes Schemmel, and René Schüffny. Wafer-scale vlsi implementations of pulse coupled neural networks. In *Proceedings of the International Conference on Sensors, Circuits and Instrumentation Systems*, 2007.

[7] Barchi F. Sviluppo software per l'ottimizzazione di spiking neural network simulations su hardware multiprocessore neuromorfico. 2015.

[8] Charles M. Fiduccia and Robert M. Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175–181. IEEE, 1982.

[9] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[10] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International journal of neural systems*, 19(4):295—308, August 2009.

[11] Norman E Gibbs, William G Poole Jr, and Paul K Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Transactions on Mathematical Software (TOMS)*, 2(4):322–330, 1976.

[12] M Herrmann, JA Hertz, and A Prügel-Bennett. Analysis of synfire chains. *Network: computation in neural systems*, 6(3):403–414, 1995.

[13] AL Hodgkin and AF Huxley. Propagation of electrical signals along giant nerve fibres. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, pages 177–183, 1952.

[14] Eugene M Izhikevich et al. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[15] Eric R Kandel, Giuseppe Spidalieri, Virgilio Perri, James H Schwartz, and Thomas M Jessell. *Principi di neuroscienze*. Cea, 1994.

[16] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[17] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[18] Javier Navaridas, Mikel Luján, Jose Miguel-Alonso, Luis A Plana, and Steve Furber. Understanding the interconnection network of spinnaker. In *Proceedings of the 23rd international conference on Supercomputing*, pages 286–295. ACM, 2009.

[19] The University of Manchester. Spinnaker.

[20] Eustace Painkras, Luis Plana, Jim Garside, Sally Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, Steve B Furber, et al. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *Solid-State Circuits, IEEE Journal of*, 48(8):1943–1953, 2013.

[21] François Pellegrini. A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. In *European Conference on Parallel Processing*, pages 195–204. Springer, 2007.

[22] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In Heather Liddell, Adrian Colbrook, Bob Hertzberger, and Peter Sloot, editors, *High-Performance Computing and Networking*, pages 493–498, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[23] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[24] Tobias C Potjans and Markus Diesmann. The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cerebral cortex*, 24(3):785–806, 2014.

[25] Pasquale Rosati, Giuseppe Caputo, and Mario De Vincentiis. *Istologia*. Edi. ermes, 1981.

[26] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, May 1969.

[27] E. Stromatias, F. Galluppi, and S. Furber. Power analysis of large-scale, real-time neural networks on spinnaker. *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages –, August 2013.

[28] G. Urgese, F. Barchi, E. Macii, and A. Acquaviva. Optimizing network traffic for spiking neural network simulations on densely interconnected many-core neuromorphic platforms. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2017.