Improving the Interconnection Network of a Brain Simulator

Jonathan Heathcote

The University of Manchester

Abstract

Many attempts to understand the vast complexity of the brain centre on simulating models of its behaviour. Conventional super-computer topologies have proven poorly suited to the communication-heavy nature of neural simulations resulting in the development of many special-purpose systems. This project aims to propose a new neural simulation architecture with a focus on interconnection topology.

Preliminary work has assessed the performance and practicality of the SpiNNaker neural-simulation architecture's topology. In addition, a possible semi-random topology is proposed to reduce the worst-case performance of the network.

The next stages of the project will develop a interconnection network simulator which will be used to evaluate and compare new interconnection topologies. This work will eventually lead to the design of a new architecture.

Contents

List of Figures

Chapter 1

Introduction

Modern computer systems are some of the most complex devices ever constructed. Current computer technologies have enabled everything from global, near-instantaneous communications via the internet to faster and more effective cancer treatments [1]. Despite this, the brain still outperforms conventional digital computers at many tasks; for example brains are able learn far more flexibly than any known computer system.

Considerable effort has been made by researchers to understand how the brain works. The small-scale operation of individual neurons in the brain is now relatively well understood but the manner of their complex interactions is still not clear. Current attempts to understand this involve modelling the behaviour of huge networks of neurons with the hope of gaining a better understanding of their collective behaviour. Such models are challenging to fit into modern computer architectures since the vast parallelism available to neurons in the brain sharply contrasts with the highly serial computing resources available today.

In this report I will elaborate on the role that modelling has on understanding the brain and in particular on the work being done to build machines to fulfil this task. I will also outline the importance of the interconnection networks in brain simulations.

1.1 Why Model The Brain?

Cutting-edge neural models can be broadly divided into two types. The first type uses simplified models of neurons in large networks. Models of this type, such as Spaun [2], are able to demonstrate a remarkable range of cognitive abilities such as memory, problem solving and pattern recognition. Spaun is notable due to its range of functional and realistic behaviours. For example, it exhibits very similar behaviour to humans in terms of the way its behaviour degrades over time due to aging or illness. It also suffers the same gradual reduction in function as random neurons die off. This level of realism means that experiments which would not be possible to carry out on real brains, such as testing hypotheses of the causes of certain illnesses, may become feasible using simulated models.

In contrast the second type of model has focused on developing high-accuracy simulations of much smaller collections of neurons to test understanding of biological processes. Such models are highly biologically plausible but do not result in the complex, high-level behaviour shown by simulators such as Spaun.

This work focuses on the first type of model as these models present a greater challenge to modern computer systems. Models of the second type, such as the Blue Brain project, are able to utilise commercial super-computers to achieve simulation speeds around one order of magnitude below biological real-time [3]. By contrast models of the first type, such as Spaun, do not easily fit current super-computer architectures and can take around two and a half hours of compute time to simulate one second of neural activity on a high-end workstation computer. §2.1 describes the basic mechanism of these neural simulators and their computational challenges in further detail.

1.2 Conventional Approaches

Conventional super-computers are designed to provide immense computational power with quadrillions (10^{15}) of numerical calculations being performed per second. Despite such impressive feats of computation, these machines often feature relatively slow interconnection networks tying the internal processing elements together. This balance of great computation and limited communication is contrary to the brain whose individual neurons offer relatively little computational power but are extraordinarily well connected.

The Blue Brain project is unusual in its use of conventional super-computers but is severely limited in network size because of this. Only around 100,000 neurons can be simulated compared with almost 100 billion in the human brain.

§2.2 goes into greater detail on current super computer architectures and their strengths and weaknesses.

1.3 The Importance of Interconnect

Given the unsuitability of traditional super-computer architectures, my research focuses on the challenge of developing new architectures for the simulation of large-scale networks, such as Spaun, which are heavily communication bound. The SpiNNaker project [4] has developed one such architecture based on a network of over one million small, low-power processors. The purpose-built interconnection network is designed to handle neural simulations of up to one billion neurons running in biological real-time. A detailed introduction to the SpiNNaker architecture is given in §2.4.

SpiNNaker's interconnection network is designed to allow efficient transmission of signals produced by simulated neurons. As the system scales up from tens to thousands of nodes, new types of interconnection technology have been introduced to keep the wiring in the system practical. §2.5 describes the technology used to simplify the system's wiring with preliminary work (§3.1) to assess the impact of these changes for neural network simulation.

Practical issues when designing interconnect, such as wiring complexity, are studied in §3.2 where the task of wiring up large SpiNNaker systems consisting of 1,200 circuit boards is tackled. This is followed by the development of an unconventional, semi-random interconnection network design which takes into account these wiring considerations in §3.3. This semi-random design of network has the potential to improve the performance of SpiNNaker with a minimal additional cost in cabling requirements.

My work will develop a new architecture intended to succeed SpiNNaker with a focus on the design of the interconnection network. §4 outlines the research I plan to undertake to reach this goal following on from my preliminary studies.

Chapter 2

Background

This chapter gives an overview of the work being done on brain simulators. It begins with a history of the models used to simulate the brain which led to the current generation of 'spiking neural networks' and the computational challenges these bring. The latest super-computer technology is then presented with an explanation of its unsuitability for neural simulation. This is followed by a discussion of the various attempts made to produce a special-purpose architecture for this task looking in particular at the highly flexible SpiNNaker architecture. The chapter concludes by examining the technology used in SpiNNaker and other super-computers to allow fast communication within these systems.

2.1 Simulating Brains

Efforts to simulate the brain focus on building approximate but representative models of its behaviour. Most current brain models attempt to capture the way that neurons behave and communicate with each other. Typically, the network is expressed as a graph of neurons which take inputs from other neurons and perform some simple computation to produce an output which is, in turn, connected to other neurons. This type of model is known generally as an Artificial Neural Network (ANN).

2.1.1 Generations of ANN

The development of ANNs can be divided up into three coarse generations, each increasing their level of biological realism [5].

The first generation of ANNs, such as the McCulloch-Pitts threshold neuron [6], consisted of testing if a simple, linear function of the neuron's inputs was above a threshold value and outputting either a 'high' or 'low' signal. The function used in each neuron and the pattern of connectivity in the network define the behaviour of the network as a whole.

It was realised that communication between neurons is not level-based but instead appears to be based on the rate at which 'spikes' are produced (or 'fired') by neurons to their neighbours. The second generation of ANNs seek to model this by representing the 'firing rate' as their output in a continuous value [7]. Once again, the network's behaviour was defined by the functions computed by each neuron and the network's connectivity.

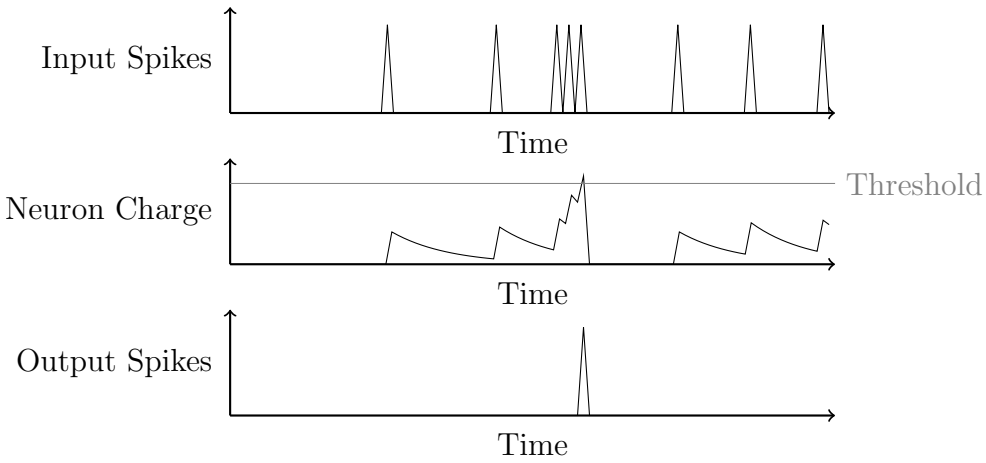


Figure 2.1: Simple leaky-intergrate-and-fire neuron example.

The third generation of ANNs extends the idea further by realising that the firing rate is not the only significant factor but the timing of the arrival of spikes is too [8]. Typically these Spiking Neural Networks (SNNs) work on the principle related to the 'leaky integrate and fire' model. Here each spike either positively or negatively contributes to the amount of 'charge' stored in the neuron (that is, the charge is integrated over time). Once the charge reaches a certain threshold, the neuron 'fires' causing a spike to be transmitted and the charge in the neuron to return to zero. Charge in the neuron also constantly 'leaks' away such that if the neuron doesn't receive any spikes its charge will eventually return to 0. This process is illustrated in figure 2.1

2.1.2 Computational Challenges

Simulating ANNs, and in particular SNNs, presents a number of computational challenges.

The first issue is that real, biological neural networks can contain billions of neurons. An adult human, for example, has around 85 billion neurons [9]. As a result, the size of biological-scale networks can be extremely large requiring vast amounts of conventional computational resource to model. Conventionally this is achieved using large, parallel computers.

The second issue is that each neuron is typically connected to thousands of others. As a result of this, huge amounts of one-to-many communication must take place between the processing nodes simulating the neurons. By contrast, conventional electronic circuits, and consequently conventional computing systems, tend to feature one-to-one and one-to-few connections as the power needed to send a signal to many places at once can be large.

Luckily most communication occurring in the brain is highly local which means that spikes do not usually need to be transmitted to neurons far away. Neurons also tend to fire at a relatively slow rate compared to electronic components with firing rates being measured in terms of hertz while modern electronics operates at multiple gigahertz. As a result time-division multiplexing can be used in order to use a single processing node or communication channel for many different neurons and spikes at once respectively.

Rank	Name	Pflops	Cores	Nodes	Topology	Interconnect	Sources
1	Tianhe-2	33.86	3,120,000	16,000	Fat-Tree	Electrical & Optical	[10]
2	Titan	17.59	560,640	18,688	3D Torus	Electrical	[11]
3	Sequoia	17.17	1,572,864	98,304	5D Torus	Electrical & Optical	[12]
4	K Computer	10.51	705,024	68,544	6D Torus	Electrical	[13, 14]
5	Mira	8.59	786,432	49,152	5D Torus	Electrical & Optical	[12]

Table 2.1: Top Five 'Top500' Super-Computers, June 2013 [15].

2.2 Super-Computers

The Top500 list [15] aims to enumerate biannually the 500 fastest super-computers ranked by their performance on the LINPACK benchmark [16]. The list offers an insight into the current state-of-the-art for high-performance computing. Table 2.1 shows the top five machines in the Top500 list released in June 2013 along with basic details of the type of interconnection involved. In this section an overview is given of the architecture of these large scale machines.

The LINPACK benchmark performs computations to "analyze and solve linear equations and linear least-squares problems" to produce a computational load representative of certain computational tasks common to scientific computing [17]. In particular it is a CPU-bound problem which attempts to measure the peak CPU performance achievable[1] but without any significant indication of the performance of the network which connects the system together [18].

Since simulation of SNNs requires a large amount of communication but relatively small amounts of computation, this benchmark is not representative of the workload of such a task. Despite the shortcomings of the LINPACK benchmark there is unsurprisingly a high degree of correlation between CPU power and interconnect performance in the best Top500 machines. The Graph500 list is a more recent, complementary ranking which uses benchmarks based on graph traversal problems [19]. Such problems rely on having efficient point-to-point communication between different parts of the system where each section of the graph resides. With the exception of Titan, which was not measured, the top five Top500 computers also sit within the top six places of Graph500 [20]. As a result, due to its maturity and wider scope, the Top500 list is discussed in this section.

2.2.1 Anatomy of a Super-Computer

Large computer systems are typically built by combining a large number of 'processing elements' in such a way that they are able to communicate efficiently. These processing elements come in many forms though the three most prevalent:

General Purpose Processor A conventional processor 'core' as found in desktop and mobile computer CPUs. These flexible devices have historically represented the vast majority of the Top500's computing power.

[1]Where CPU Performance is measured in Petaflops: how many quadrillion (10^{15}) floating point operations can be performed per second.

Graphics Processing Unit (GPU) A specialised processor which is able to efficiently perform the same operation across a large number of data elements simultaneously. The Titan super-computer notably makes extensive use of GPUs [11].

Accelerators Increasingly other forms of compute resource, such as the Intel Xeon Phi accelerator in Tianhe-2 are being used. The Xeon Phi contains 57 medium-sized general purpose cores which can be used to complement a conventional multi-core system [10].

A number of these individual processor cores are then combined together to create a single 'node' in the system. Typically the cores within a node are able to communicate relatively cheaply while messages to remote nodes must traverse a slower system-wide interconnection network.

2.2.2 Topology

The topology of the system-wide interconnect has a great influence on the performance of a super-computer. Typically the topology of these networks is such that communication between near-by nodes is cheap and does not compete with distant nodes for communication resources. The top five machines in the Top500 list achieve this using the 'fat tree' and 'torus' topologies described below.

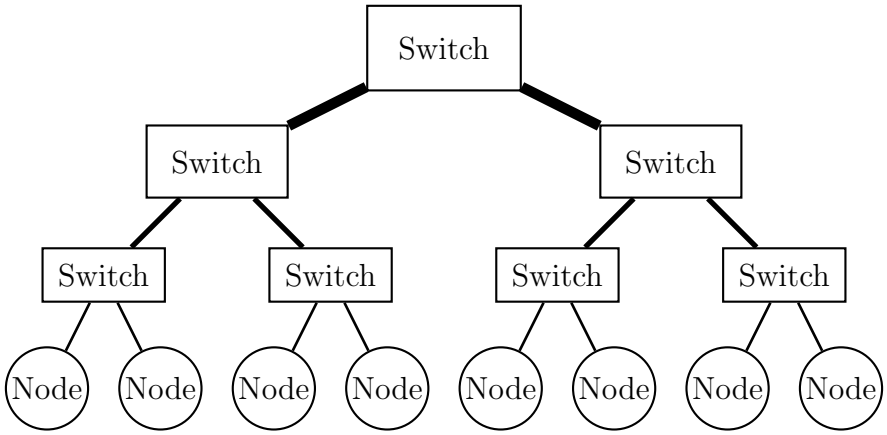
2.2.2.1 Fat Trees

In a basic fat tree, nodes are connected in a tree structure to switches which in turn are connected to further levels of switches (figure 2.2a). Connections higher in the switch hierarchy are connected via links of increasing bandwidth to avoid the bandwidth bottleneck around the root node. For a node to communicate, it sends its message to its parent switch which forwards the packet through the tree to its destination.

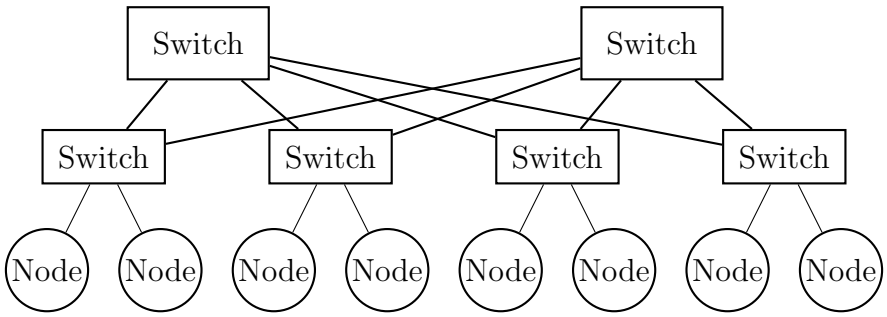
In practice, such as in large machines like 'Tianhe-2', it is not possible to build a switch with the required bandwidth of the higher nodes of a fat tree. In addition, the root switch is a potential single point of failure for the whole system. As a result, folded Clos networks are often used instead and have become synonymous with the fat tree topology. As shown in figure 2.2b, traffic is split between two top-level switches reducing the bandwidth requirement for each. In addition, if a top-level switch fails messages can simply be routed using the other switch.

Fat trees give small maximum hop-counts ($O(\log N)$ with respect to the number of nodes in the system) to send a message from one node to any other. They also allow near-by nodes to communicate in a small number of hops. Unfortunately, such topologies depend on high-radix switches which can connect many nodes simultaneously. Tianhe-2, for example, has thirteen 576-port switches based on a custom router chip. The cost of using high-radix switches is typically that of increased latency. In the case of Tianhe-2, the latency of a message broadcast to all nodes in the system is around 9μs or about 19,800 CPU[2] cycles [10].

[2]Intel Ivy Bridge CPUs running at 2.2 GHz



(a) Basic fat tree links.



(b) Folded Clos network.

Figure 2.2: Fat tree topologies. Thicker lines represent higher bandwidth links.

To allow multiple simultaneous computation tasks to share the super-computer's resources, fat trees can be trivially partitioned by allocating whole sub-trees to a given task.

2.2.2.2 Tori

The most common topology which features in all but one of the top-five is the torus (also known as a k-ary n-cube). In this topology, nodes are arranged in a n-dimensional mesh. Nodes at the extreme edges of the mesh are connected together to form a torus. In the 2D case this can be visualised as in figure 2.3. Starting with a regular 2D mesh (figure 2.3a), the top- and bottom-most nodes are connected together to form a tube (figure 2.3b). Then the left- and right-most nodes of the tube are connected together forming a torus (figure 2.3c). Though harder to visualise, this process generalises to toruses of higher dimensions.

Each node is able to communicate directly with its immediate neighbours in each dimension, that is above, below, left and right in the 2D case. More distant nodes are able to communicate by forwarding messages via intermediate nodes.

Because nodes in a torus only communicate directly with their neighbours, the switches required in each node may be of a low radix compared to those in the fat tree. For an n-dimensional torus, each node requires a radix $2n + 1$ switches independent of the total number of nodes in the system.

Compared with a fat tree, however, a greater number of hops is required in the worst case where two nodes are distant. In a torus k nodes long in each of n dimensions the worst case path length is $\frac{kn}{2}$ [21]. As a result, though the individual hops are less expensive, the time taken to broadcast a message to all nodes in a large Blue Gene/Q supercomputer such as Sequoia is 17 μs compared to 9 μs for the same operation in Tianhe-2 [22].

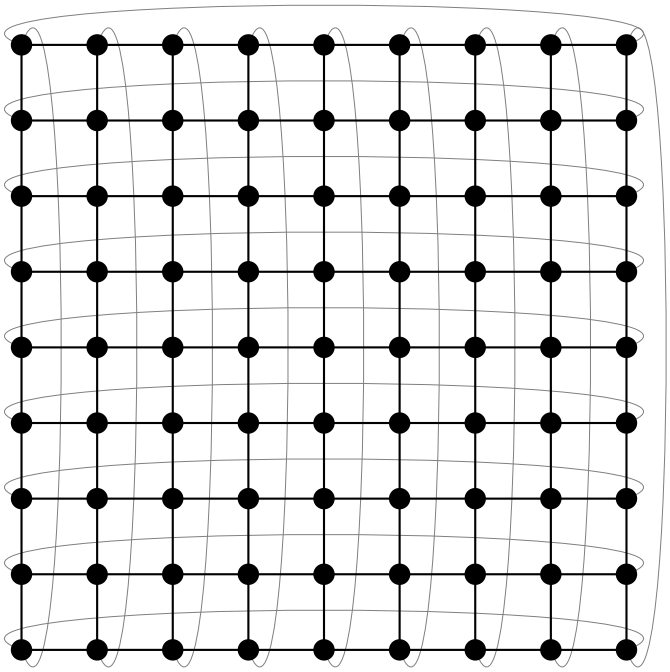
Higher dimensional tori are also easily partitioned into a number of lower-dimensional tori or meshes to allow machines to be shared between many independent computing tasks [14, 23].

2.2.3 Interconnect

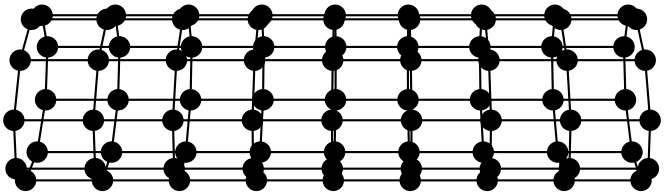
The physical links responsible for connecting machines are another important factor in the performance achievable by a given interconnection system. The current state-of-the-art techniques fall into two categories: electrical ('high-speed serial') and optical transmission.

Electrical transmission technologies are generally much cheaper than optical for short distances and lower bandwidths. As a result connections between physically neighbouring nodes are almost universally connected via such links. In systems such as Blue Gene/Q and Tianhe-2 optical links are used to connect between different cabinets in the system [10, 12]. These optical links are able to carry the equivalent of many electrical signals over longer distances at the expense of more complex hardware requirements for transmitters and receivers.

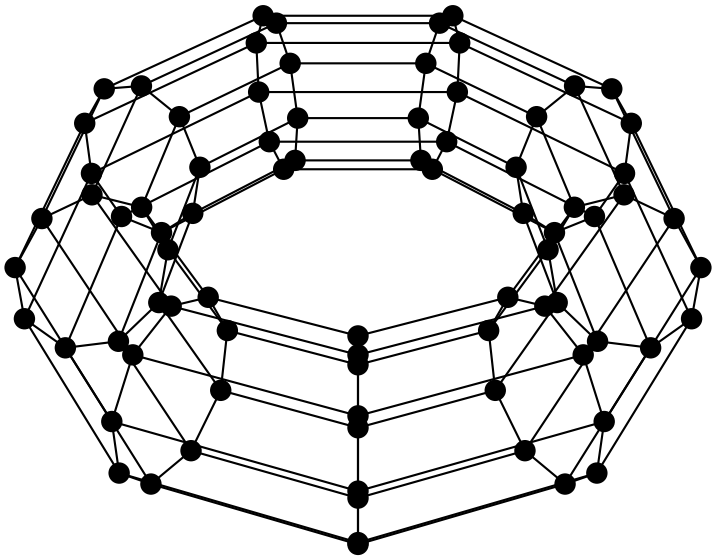
The topology of a network can greatly influence the difficulty of physically connecting nodes arranged in cabinets. The use of cabinets essentially map the physical nodes into an approximately 2D sheet. As a result, for torus networks of more than two dimensions,



(a) Mesh (Grey lines show wrap-around connections added in a torus)



(b) Rolled into a tube



(c) Bent into a torus

Figure 2.3: Transformation of a mesh into a torus.

links which are physically short in higher dimensions can result in long wires between distant cabinets. For hierarchical networks such as fat trees, long wires arise from the need to connect together distant parts of a system at the higher levels of the hierarchy. Long wires are both more difficult to install and require more expensive link technologies. The preliminary work described in §3.2 explores these issues in further detail.

2.3 Hardware Neural Simulators

Current super-computers are heavily focused on computation-heavy tasks as is apparent from the use of hundreds of thousands of high-end processor cores in the Top500's top five machines. The Blue Brain project [3] makes use of such computationally powerful machines to simulate relatively small ANNs of tens of thousands of realistic neurons. Large ANNs using the less realistic models of neuron behaviour change the balance between computation and communication. Requiring relatively little computation for each neuron but needing vast amounts of communication, conventional super computers are a poor fit and, as a result, alternative architectures have been developed for neural simulation.

2.3.1 Electrical Technology

There are three distinct approaches being taken to designing (electrical) hardware for neural simulation. One is to use analogue electronic components for the entire system, another is to mix analogue and digital components and a final approach is to use traditional digital components. These three approaches and their merits are described below.

2.3.1.1 Analogue

Though analogue technology has long been out of favour for general purpose computing, it has been suggested for neural simulation as it is ultimately designed to model an analogue system: the brain. Neural models are highly fault-tolerant and do not assume that values will be calculated or communicated precisely. Modern digital computers, by contrast, have rigid requirements for the precision of values and so dedicate vast amounts of hardware and energy to guaranteeing precision, which is unnecessary in this case, making analogue simulators an attractive approach.

A wide range of techniques for implementing the required functions in analogue circuitry has been proposed [24, 25, 26, 27]. The analogue circuits used are often simple and replace more power-hungry general purpose processors [28]. Even though the lack of precision of analogue circuits is not a problem for neural simulation, the lack of consistency is. The same analogue circuit may have widely different characteristics in one part of the chip compared to another due to variations in the silicon wafer. In addition circuits must tolerate changes in temperature and voltage, a task which is substantially more challenging for analogue circuits compared to their digital counterparts. As a result, current successful implementations such as the MIT Silicon Synapse have been limited to small parts of a neural network, a single synapse connecting two neurons in this case [29].

Finally, analogue systems make assumptions about the type of neural models that will be simulated limiting them to simulating only a handful of similar models. As a result,

the current generation of analogue simulators lag behind the state-of-the-art in terms of ease-of-use and flexibility.

2.3.1.2 Mixed Mode

So-called 'mixed mode' systems have been designed such as Neurogrid and BrainScaleS which combine analogue computational components with digital communication [30, 31]. The use of digital communication avoids issues with inconsistencies across the chip while also simplifying routing of spikes which can then be completed using conventional digital methods.

While offering improvements over purely analogue systems, many of the same drawbacks with variability and flexibility still apply.

2.3.1.3 Digital

Various groups have developed custom, digital, on-chip neural simulators [32, 33, 34, 35]. While successful in allowing relatively large numbers of neurons to be simulated, around 1 million in the case of SP^{2}INN, along with the analogue and mixed mode designs above, these approaches are fundamentally restricted to only the models the designers originally intended [35]. Due to the lack of flexibility and increasing costs of designing custom silicon, researchers, such as those behind SP^{2}INN, have moved away from this approach.

FPGAs (Field Programmable Gate Arrays) are, essentially, chips which can be electrically reprogrammed with custom logic circuits. They are now widely used in place of custom chips for performance-sensitive, highly specialised tasks as well as prototyping and development of conventional chips. Work using FPGAs to simulate neural models such as [36] has allowed around half a million neurons with realistic numbers of connections between them to be simulated. Despite being considerably cheaper and far more flexible than completely custom hardware, FPGA development is still much slower than developing for general purpose CPUs and so still yields relatively inflexible systems.

Finally, architectures based on general purpose digital processors are able to provide a great degree of flexibility at the expense of greater overhead[37]. The SpiNNaker project, described in detail in the next section, has developed an architecture which combines many low-power processors which run neural simulations in software.

2.3.2 Topologies

Though various topologies have been used in hardware neural simulators these largely boil down to those based on to tori and trees. Unlike those in super-computers, the granularity of these networks tends to be higher reflecting the increased focus on communication over computation. This subsection examines the topologies of two leading mixed-mode neural simulators, Neurogrid and BrainScaleS.

2.3.2.1 Neurogrid

The largest prototype Neurogrid system is able to support around 1 million neurons in biological real-time using 100,000 times less power than a conventional super-computer. Its architecture, at the highest level, consists of a small number of Neurogrid chips (16

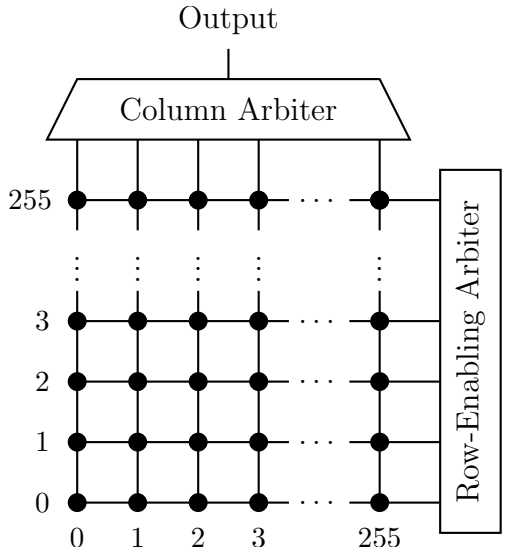


Figure 2.4: Neurogrid chip topology showing the mechanism used to arbitrate the chip output port.

in current prototypes) arranged as leaves in a binary-tree where spike events are routed to other chips [30].

Each Neurogrid chip then contains a 2D grid of 256×256 analogue neuron models as shown in figure 2.4. When a neuron produces a spike it asserts a signal sensed by the row-enabling arbiter. This then selects a single row which is then allowed to assert a signal on a column wire. The column arbiter then selects a single neuron from the enabled row which is then output by the chip [38]. When a spike arrives for a neuron in the chip, a similar process occurs to single out the neuron for which it is destined [39].

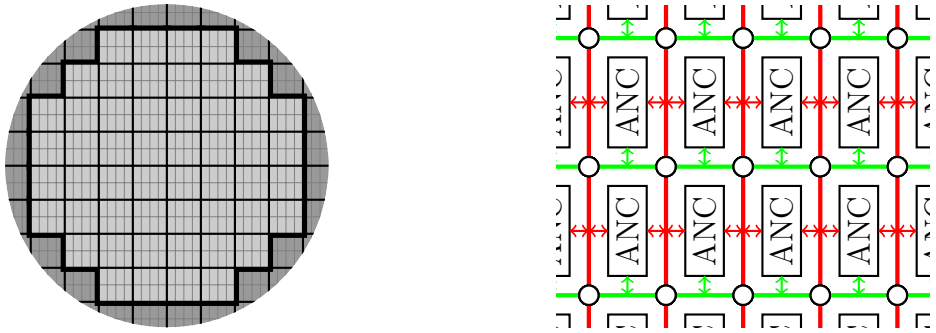
The on-chip topology is cheap to implement as row and column connections are simple wires and the controlling logic is relatively simple. Unfortunately, it does not exploit the available parallelism in the system allowing only one spike to be emitted at once. In addition, spikes destined for other neurons near-by in the chip are just as expensive as spikes between distant neurons on the same chip.

Off-chip, spikes traverse the binary tree structure which is easily scaled to larger systems with the limit being the number of bits available for neuron identification.

2.3.2.2 BrainScaleS

The BrainScaleS project makes use of wafer-scale integration where multiple reticles produced on the same silicon wafer are connected together rather than split into separate dies. A system built from a single wafer is able to simulate a network of around 200,000 neurons at up to 10^5 times biological real-time while using 6,000 times less power than a conventional super-computer [40].

As shown in figure 2.5a, each reticle, outlined in black, on the wafer contains eight Analog Neural-Network Chips (ANCs), outlined in grey, which can be programmed to model up to 254 neurons (depending on the number of connections to other neurons) [41]. There are two types of interconnect in the system named 'level 1' (L1) and 'level 2' (L2) which are described below.



(a) BrainScaleS wafer with usable reticles shown in light grey with eight ANCs each.

(b) L1 interconnect between ANCs with sparse cross-bar switches at intersections.

Figure 2.5: Overview of the BrainScaleS topology.

The L1 network is a mesh network (that is, a torus without wrap-around links) which connects all the ANCs on the same wafer together as shown in figure 2.5b. The links of the mesh are circuit-switched meaning that connecting a set of points on the mesh 'uses up' all the wires along the way preventing others from using them.

Since each link in the mesh may be useful or required by connections between more than one set of ANCs, the links consist of 64 or 128 separate 'lanes' for horizontal and vertical links respectively [42]. Connecting up a particular set of ANCs now uses only one lane out of each link allowing many more connections to pass along the same link. At intersection points in the mesh, each lane can be configured to be disconnected allowing the two parts to be used for separate purposes. Sparse cross-bar switches are used at the intersection points to cheaply connect a subset of the lanes in the intersecting links.

The mesh topology of this network allows cheap local communication as is commonly found in neural networks. In addition circuit switched networks offer extremely low latency, low-power connections between ANCs compared with a similar packet switched network as used in super computers. This performance comes at the expense of flexibility as lanes, once allocated, cannot be used for any other purpose even when idle. This can make changes to the routing of the network during runtime very difficult [21]. This network is also limited by the size of the wafer which presents a severe scaling limit.

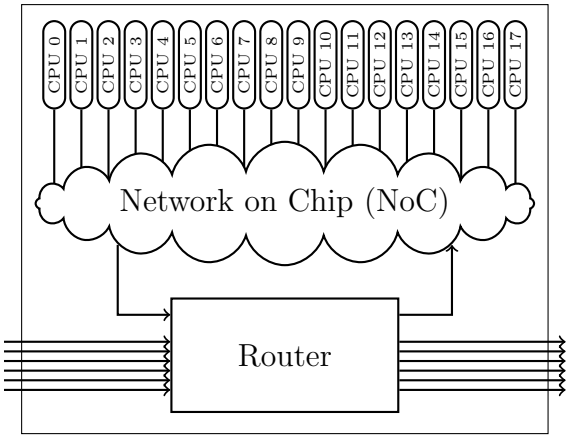
The L2 network in BrainScaleS is facilitated by connecting each ANC to a custom chip which translates communications into UDP/IP network packets which are sent via 10 gigabit Ethernet links to a commodity data-centre network switch [41]. This network suffers higher latencies than the L1 network but allows signals to be routed much more flexibly and offers the possibility of expanding the system to multiple wafers.

The use of a central switch essentially results in a star topology preventing significant scaling of the system without introducing further hierarchy into the network.

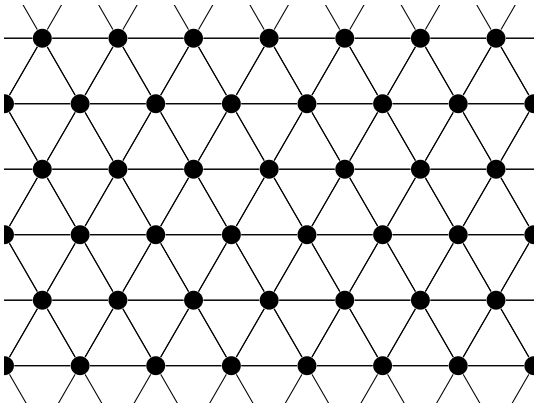
Though well suited to the size of current BrainScaleS prototypes, both the L1 and L2 network topologies represent a challenge to further scaling.

2.4 SpiNNaker

The SpiNNaker project is developing a super-computer architecture designed for running real-time simulations of large networks of SNNs. In particular, it is designed to be flexible,



(a) SpiNNaker chip



(b) Connections between chips (dots) with wrap-around connections not shown

Figure 2.6: Overview of the SpiNNaker architecture.

making as few assumptions about the function of individual neurons as possible [4].

To achieve this flexibility SpiNNaker is planned to combine over one million energy-efficient, general purpose, mobile phone grade CPUs with a custom interconnect network designed to tackle the communication bound problem of simulating large networks of simple neurons. The machine is planned to support networks of around one billion (10^9) spiking neurons (around 1% of a human brain) in biological real-time.

Due to the machine's flexibility and novel interconnect it presents many opportunities for experimentation and so has been the focus of much of the preliminary work done so far. The rest of this section describes SpiNNaker highlighting areas relevant to the preliminary work and final project.

2.4.1 Architecture

SpiNNaker is made up of a 2D toroid[3] of chips where each chip connects to its six neighbouring chips as shown in figure 2.6b. The system can be flexibly extended by simply enlarging the toroid and adding more chips. The size of the network is only limited by the size of address fields used to route spikes around the system.

Each SpiNNaker chip contains 18 low-power ARM processor cores connected together via a network on chip (NoC) as shown in figure 2.6a. Each core has a small amount of private memory and has access to a larger, chip-wide memory (not shown). Finally the chip contains a router responsible for sending and receiving messages from the six neighbouring chips as well as forwarding messages destined for another chip.

2.4.2 Routing

The on-chip router is table-based meaning that routing decisions are taken based on a set of predetermined choices loaded into a memory before simulations begin. The router

[3]A toroid, in this context, is the generalisation of a torus where nodes are connected to more than $2n$ neighbours where n is the number of dimensions.

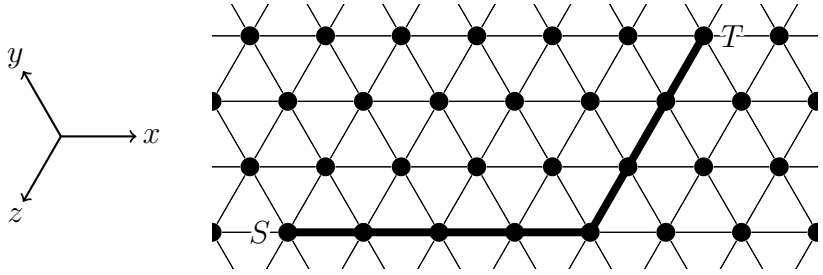


Figure 2.7: Dimension order routing in SpiNNaker showing a path from node S to T.

operates on individual packets of data which are either 40 or 72 bits in length. These packets are short compared with those commonly used in super-computers which can be several kilobytes long. Such networks are typically designed to transfer larger blocks of data, possibly many kilobytes long. SpiNNaker, however, is designed to primarily deal with SNNs where only the spikes' existence needs to be transmitted requiring little or no associated data. The system supports four different kinds of packet:

Point-to-Point (P2P) Addressed to an individual chip and will be passed to the core arbitrarily selected by the chip as the 'monitor' core. Packets are routed using a table.

Multicast (MC) Sent by a single core and delivered to a predetermined set of cores in the system. Packets are routed according to their source address and do not explicitly store the destination cores. Packets are routed using tables.

Fixed Route (FR) Automatically routed to a predefined central point in the system. Used, for example, to collect diagnostic information back to the host machine. By not requiring an address a larger payload can be attached.

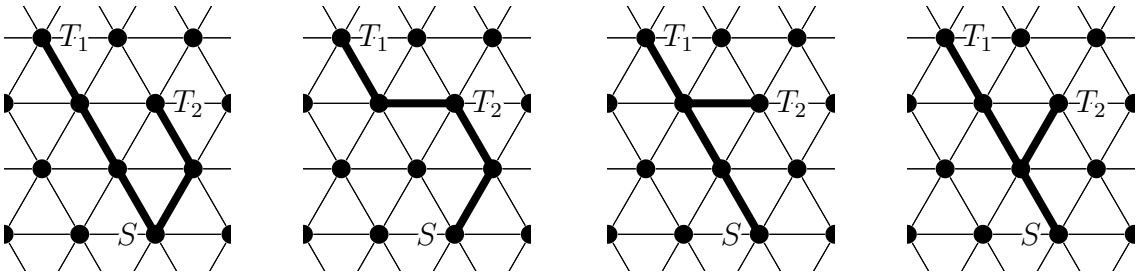
Nearest Neighbour (NN) Routed to one or all of the immediate neighbours of a chip without the need for a routing table. On system start up routing tables have not been initialised and so NN packets are used to initially load the tables.

2.4.2.1 Routing Scheme

Table based routing allows a wide array of possible routing schemes to be implemented conditional to two key constraints. First, the route taken by a packet with a particular source/destination address will always take the same path through the system. This is known as deterministic oblivious routing as the scheme is unaware of the system's state and cannot react to load imbalance. It is worth noting that schemes still exist which can improve load balance in the general case [43].

Current versions of the routing software use simple dimension order routing. Here, the three axes along which packets can travel are considered to be three 'dimensions'[4]. Packets travel along a given dimension until they can get no closer at which point they move onto the next dimension as shown in figure 2.7. Such paths are easily computed as shown in [44] and are minimal in the number of hops required.

[4]The three dimensions in SpiNNaker's toroid are not orthogonal as would be the case in a conventional torus network. This is the cause of some of the slightly unintuitive properties of the topology.



Route	Router Entries	Total Hops	Hops to T_1	Hops to T_2
(a)	4	5	3	2
(b)	5	4	4	2
(c)	4	4	3	3
(d)	4	4	3	2

Figure 2.8: Multicast routing examples from S to T_1 and T_2.

2.4.2.2 Multicast

Multicast packets allow SpiNNaker to handle the efficient distribution of spikes from heavily connected neurons in the system without requiring a unique packet to be sent to each destination. Instead a single packet is transmitted which is able to 'fork' to reach multiple processors later in its journey.

Unlike P2P packets, for which the route for every possible chip address (16 bits) can be stored in a routing table of around 25 kilobytes, MC packets are routed based on a 32 bit key which uniquely identifies the neuron which fired. Exhaustively storing the route for every possible 32-bit key is not possible since 24 bits are required for each entry thus requiring a 12 gigabyte routing table.

Instead of an exhaustive table, one with 1,024 entries is used. The router looks up the 32-bit key of MC packets and, if a matching entry exists, uses that entry to decide where to route the packet. If no entry exists, the packet is forwarded to the link physically opposite the one through which it entered. As a result, the routing table only requires an entry at the entry point into the network, where the packet changes direction and where it is to be delivered to a core.

Since the route for messages with a given key will only pass through or change direction in a small number of places, most routers in the system won't need to include an entry in their routing table. As a result this small size of routing table is hoped to be adequate.

In multicast routing, the choice of where a route should fork can have a great impact on system load, packet latency and routing table entry usage. Figure 2.8 shows several examples of valid multicast routes which fork at different points, each with differing trade-offs.

As well as guaranteeing short paths and that no routing table requires more than 1,024 entries, the multicast routing scheme chosen must attempt to ensure an even load-balance by avoiding over-using certain links. Various heuristic approaches based on extensions to Lee's algorithm have been proposed for future versions of the routing software[45].

2.4.3 Hardware Abstractions

The largest SpiNNaker system is planned to contain 1,036,800 cores spread over 57,600 chips. These chips are split up on 1,200 circuit boards of 48 chips each as shown in figure 2.9.

Each board is equipped with a number of high-speed links, six of which are used to connect the board's chips to their neighbours on other boards. The other links are reserved for connecting other I/O such as silicon retinas and robotic platforms [46]. An Ethernet connection is also present to provide a simple, low-bandwidth link to an external host system.

Groups of twenty-four boards are then placed in racks such as in figure 2.10 which are in turn placed, five-high, into ten cabinets to complete the largest planned machine. Further details of this arrangement are given later in §3.2 where preliminary work on wiring schemes for the machine are described.

2.4.4 Connecting Boards Together

Logically, the chips on each circuit board are laid out as shown in figure 2.11. Touching edges represent chip-to-chip connections which uses a delay insensitive, parallel '2-of-7' communications scheme similar to that used in the on-chip network requiring 16 wires per link. If this technology was used to connect boards together 768 wires would be required. This would be prohibitively expensive requiring highly specialised cables and connectors. Instead, an alternative technology, known as high-speed serial, is used which replaces the 768 wires with only 24 which can be carried by cheap, commodity cables. This technology allows eight board-to-board connections to share a single cable grouped as shown in the figure.

To construct a toroid, at least three boards must be combined as shown in figure 2.12a (an arrangement known as a threeboard). Figure 2.12b shows how the threeboard arrangement can be turned into a sheet which in turn can be turned into a toroid as shown earlier in figure 2.3.

Arbitrarily large systems can be produced by repeating the threeboard pattern to create larger toroids.

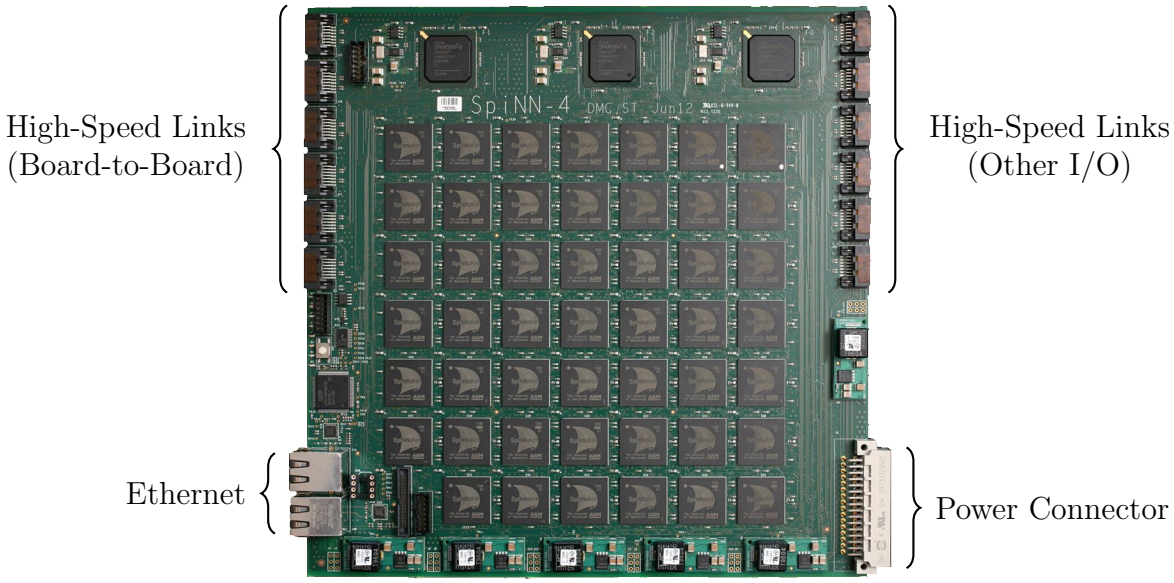


Figure 2.9: 48-chip SpiNNaker circuit board.

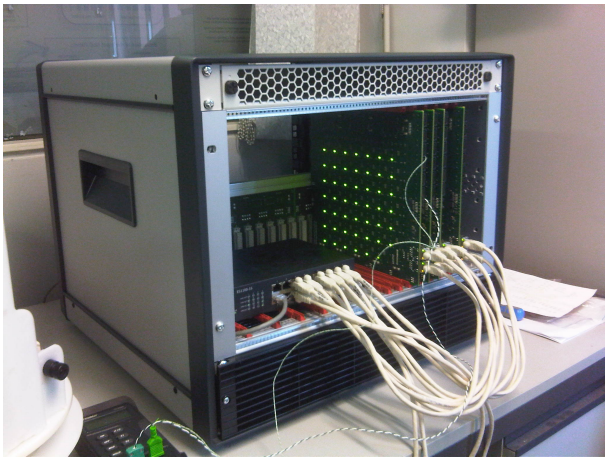


Figure 2.10: Partially populated rack of SpiNNaker boards.

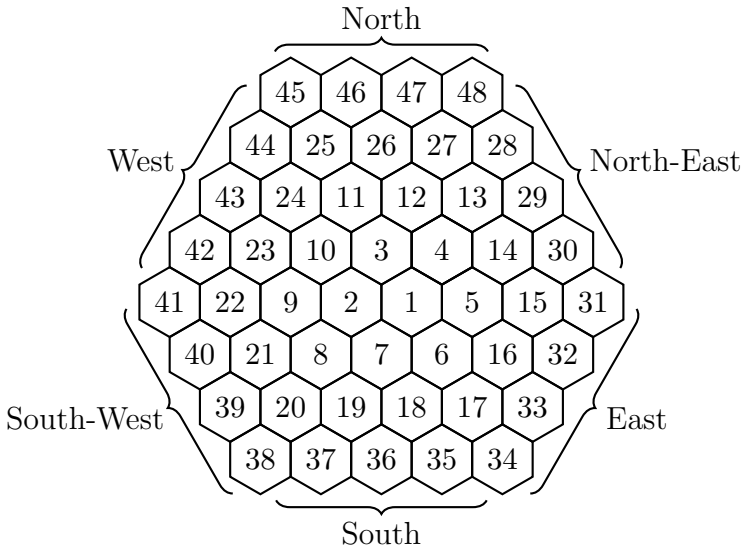
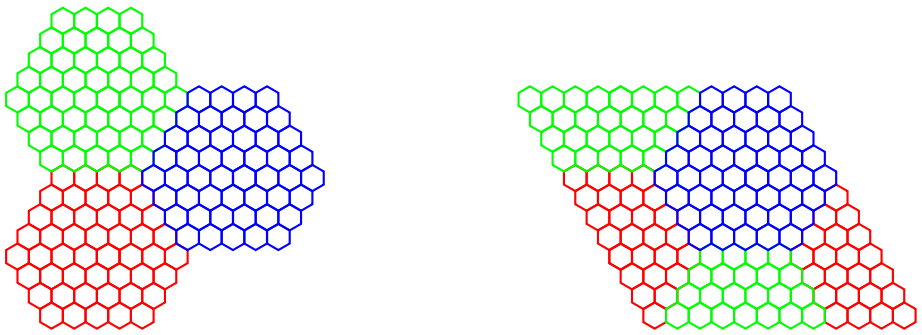
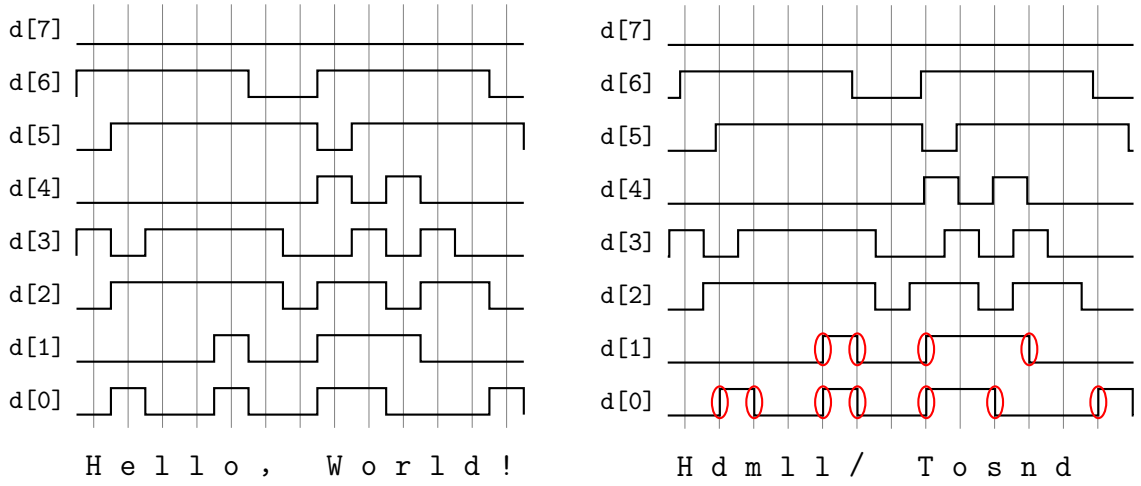


Figure 2.11: Logical arrangement of chips on a circuit board.



(a) Threeboard (b) Sliced into a sheet

Figure 2.12: A 'threeboard', the minimal configuration of boards yielding a toroid.



(a) No skew

(b) With skew

Figure 2.13: Parallel signalling example.

2.5 High-Speed Serial

High-speed serial is the general name for a technology which allows high-bandwidth links to be constructed using only a few electrical wires. These links are the basis of many widely used technologies ranging from S-ATA, used for attaching consumer-grade hard disks to computers [47], to the InfiniBand interconnect designed for super-computers [48].

Compared to older technologies such as simple parallel buses, high-speed serial links require more complex hardware to implement and so tend to be reserved for off-chip connections. They are the basis of many current Top500 super-computer interconnects as well as SpiNNaker's board-to-board links.

Traditionally, signals between components in a system have been carried by a set of parallel wires. Each wire carried a single bit allowing multiple bits to be transmitted at once. Figure 2.13a shows eight parallel signals being used to transmit a message. The electrical signal being sent down each wire is sampled at the tick of a clock (the vertical lines in the figure) and the eight bits are interpreted as an ASCII character.

In practice each of the wires carrying the parallel signal will have differing electrical properties due to imperfections during manufacturing such as slight variations in length. These differences result in the signals taking different times to travel along the wires, an effect known as skew. For a long time the skew remained insignificant compared to the period of the parallel signal. As clock speeds increased, however, skew started to become significant enough to cause some of the parallel signals to arrive so far apart relative to the clock that some would be incorrectly sampled by the receiver resulting in transmission errors such as in figure 2.13b.

Since wiring technology has been unable to improve fast enough to keep skew acceptably small for all but the shortest connections, parallel signalling has been replaced with high-speed serial signalling. Here, bits are sent one after another as shown in figure 2.14. Because there is only one signal, the problem of skew is eliminated.

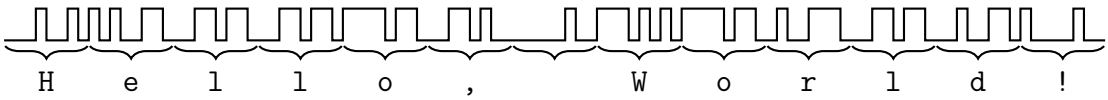


Figure 2.14: Serial signalling example.

Chapter 3

Preliminary Work

The preliminary work conducted so far aims to explore some of the practical issues faced when designing interconnection networks. The work focuses on the SpiNNaker architecture whose advanced state of development makes it an appropriate target for realistic experimentation.

The first section describes an experiment conducted to determine the way in which the choice of technologies on which an interconnection network is built can affect its performance. As well as performance considerations, this choice also has implications for the practicalities of constructing real systems. The second section describes work which studies the practical issues faced when wiring up SpiNNaker's chosen interconnect. The chapter concludes with an experiment to test a simple extension to the SpiNNaker interconnect topology. The proposed topology uses semi-random links to improve its performance while still accounting for the practical issues involved in assembly.

3.1 SpiNNaker Interconnect Modelling

An important factor in designing interconnection networks is the technology used to implement the links in the network. These choices influence the network's performance as they determine the costs involved in sending a message across a link. This in turn can, for example, affect the way packets need to be routed to make optimal use of the system's resources.

At the chip-to-chip level, the SpiNNaker system is homogeneous with identical links connecting chips to their (also identical) neighbours. Unfortunately, this homogeneity is broken when signals need to cross between boards. The board-to-board links concentrate multiple chip-to-chip signals onto a small number of high-speed serial links to reduce the cost of wiring.

Though the board-to-board connections are logically transparent, they inevitably result in some additional latency at the board boundaries. Figure 3.1 shows a high-level schematic of the board-to-board links. At both ends of the link, additional processing and buffering is required. Each link must collect packets from the attached chips into frames which are then sent across the high-speed serial link, requiring buffering at each stage. All of these steps incur additional latency over a direct connection between two chips.

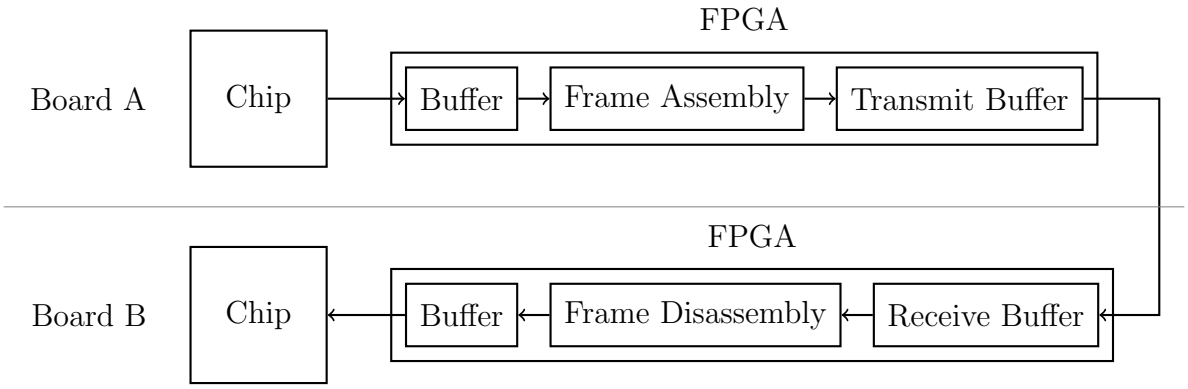


Figure 3.1: Simplified schematic of the processing stages encountered by a packet crossing between boards connected via a high-speed serial link.

The real-time nature of SpiNNaker's simulations means that the latency of packets in the system can be critical to the simulation's correctness. If a packet takes too long to be delivered this counts as a failure of the system. Indeed, packets travelling across the entire system may have to pass through a number of these links potentially incurring a high latency penalty.

Because latency between pairs of chips is no longer uniform, this violates the assumptions made by the routing schemes currently in use. The cost of this mistaken assumption is not known and one of the aims of this work was to measure its impact.

3.1.1 Simulation

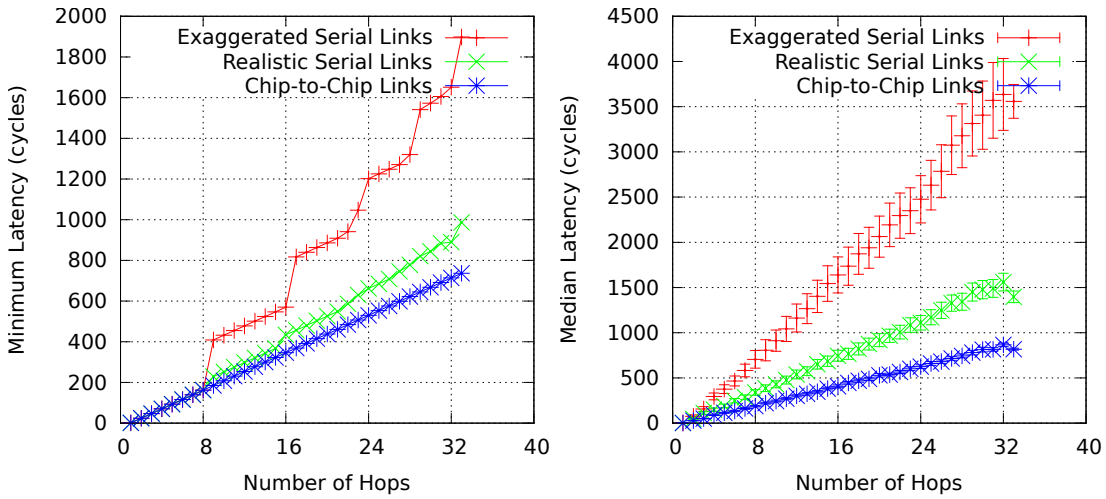
To study the effects of the board-to-board links, a simplified simulation of SpiNNaker's interconnect was built. The simulated model was based on the work presented by Navaridas et al. in [49] but using an alternative simulator written in Python to speed up development. As the high-speed serial links are under active development, they have been modelled using the expected latency values for the completed system.

3.1.2 Results

Simulations of systems containing a toroid of 4×4 threeboards (48 boards, 2,304 chips) were performed under various conditions.

3.1.2.1 Packet Latency

To test the effect of board-to-board links on packet latency, the network was simulated with each chip generating a packet every CPU cycle, with a probability of 1%, to uniform-random destinations. The latency added by the serial links can be seen in figure 3.2a where the minimum packet latency is shown against the number of links (hops) used by a packet. The system is also shown as if using only chip-to-chip links, with realistic high-speed serial links and with high-speed serial links with exaggerated latencies (for illustrative purposes) as board-to-board links.



(a) Minimum latencies.

(b) Median latencies, error bars = 0.1 × Inter-Quartile Range.

Figure 3.2: Packet latencies for various types of board-to-board links.

Steps can be seen every time the number of hops passes a multiple of 8, the number of consecutive chips in any single dimension on a board, after which a board-to-board link is crossed causing increased latency. This step change can be seen clearly in the exaggerated links but is also visible in realistic links.

An extra step appears at 28 hops, the cause of which is not currently known by the author.

The median latency of an n-hop path increases smoothly as shown in figure 3.2b as the probability of crossing a boundary increases with the number of hops carried out. From the gradient of these lines it can be seen that the realistic serial links result in an 80.4% latency overhead.

3.1.2.2 Naïve Routing Effects

The routing scheme used by current SpiNNaker simulations is based on dimension order routing. This scheme assumes that all 'hops' take the same amount of time in order to generate routes with minimal-latency. This assumption breaks down in heterogeneous systems.

Figure 3.3 shows how the latencies vary across an idle system from a single point. It can be seen that where board boundaries (shown in white) are crossed there is a general increase in latency. Because of this, the contours are visibly distorted from the expected hexagonal shape. This is particularly visible at the bottom left and top right of the figure.

A step in latency is also visible within individual boards as can be seen in figure 3.4. Here there are clear edges where dimension order routing traverses the dimensions in an order which incurs an extra board crossing. This effect is visible as a step-change in latency along the diagonals of the upper-right boards.

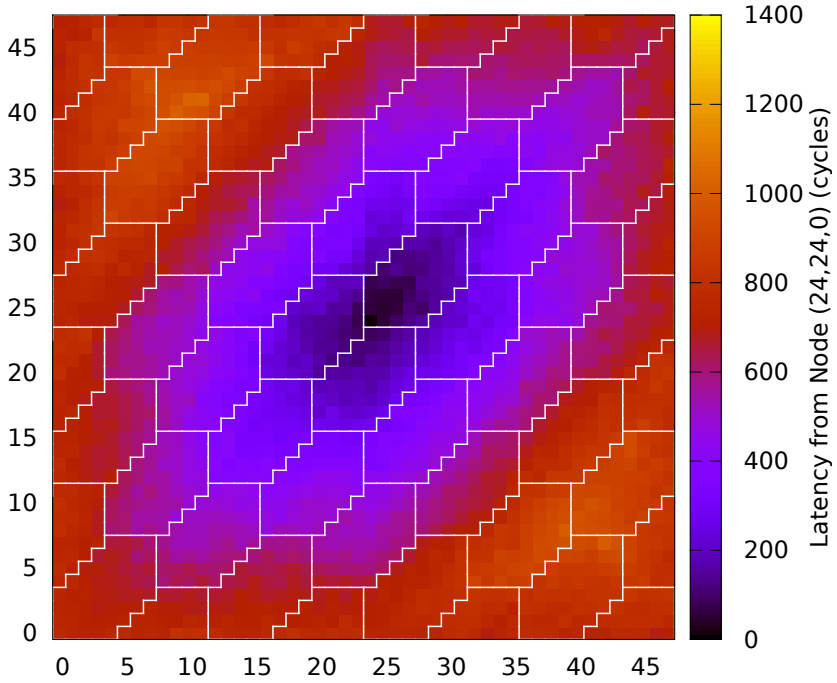


Figure 3.3: Heat-map of average packet latency to each chip from the central chip. Note: a skewed perspective is used.

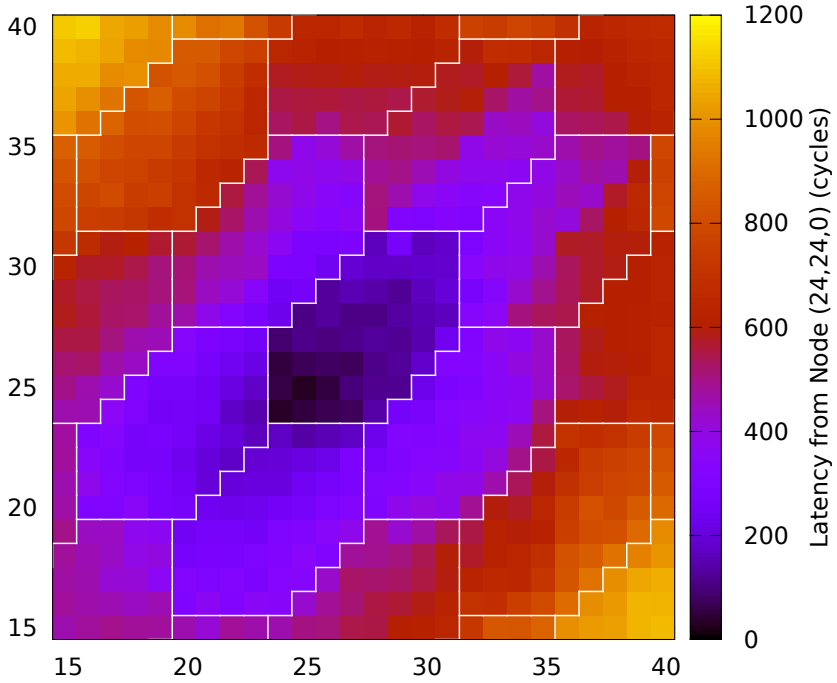


Figure 3.4: Latency to each chip from the central chip (24,24) to a section of a 48×48 system. The serial latency in this system has been exaggerated to aid visibility but the effect is still present with realistic latencies.

3.1.3 Conclusions

Though at first sight the 80.4% latency penalty seems severe, for SpiNNaker's target of SNNs running at 1 ms time steps this is not fatal. The new median latency is still only around 7.5 μs for the longest possible path, very much smaller than 1 ms deadline. For much the same reason, the effects of naïvely routing packets should also not affect current simulation schemes significantly.

For other more latency sensitive applications such as finer-grained neural simulations, the increase in latency due to poor routing choices may still be problematic.

It is worth noting that the simulations carried out were limited in size, duration and traffic complexity by the performance of the simulator. An improved simulator would allow larger models to be simulated using more realistic networks. One possible approach is to extend the simulator used by Navaridas et al. and this is discussed in §4.1.2.

3.2 Wiring-Up Large SpiNNaker Machines

One of the practical constraints on the topologies is the use of wires to connect parts of the system together. Chief amongst these concerns are the following:

Wire Cost The choice of cabling used has a direct impact on the financial cost of the wiring. Higher quality materials or increased numbers of wires can substantially drive up the cost of the cable and connectors required.

Wire Length Long wires have higher capacitances than short ones limiting the rate at which data can be transmitted.

Wiring Complexity Ultimately the system will need to be assembled (usually) by hand so keeping wiring patterns simple is important to prevent wiring mistakes.

The SpiNNaker interconnect topology was studied to confirm the existence of a practical wiring scheme which satisfies all of the above properties. The principles of the machine's construction suggested by Davidson [50] and Furber [51] were used as the basis for a tool used to model and experiment with possible configurations. This section explains the mapping developed and how it meets the above requirements. Finally, the section concludes with a description of the tool and the future work which remains.

3.2.1 Reducing Wiring Length

In figure 3.5, touching boards are connected and the lines represent wires which complete the toroid by connecting opposing edges. While the majority of wires are very short (they connect two boards which are side-by-side), some wires must cross the entire system's length. To remove long wires from a toroid network, the network can be folded [21]. An example of this process is shown for a simple ring-network (a 1-dimensional toroid) in figure 3.6.

This process is generalised to SpiNNaker's boards as shown in figure 3.7. The first step (figure 3.7a) transforms the rhombus-like arrangement of boards into a rectangle which is more easily folded.

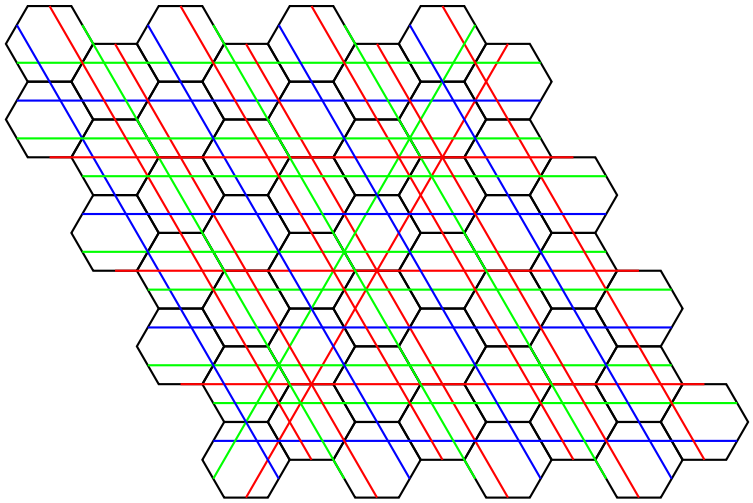
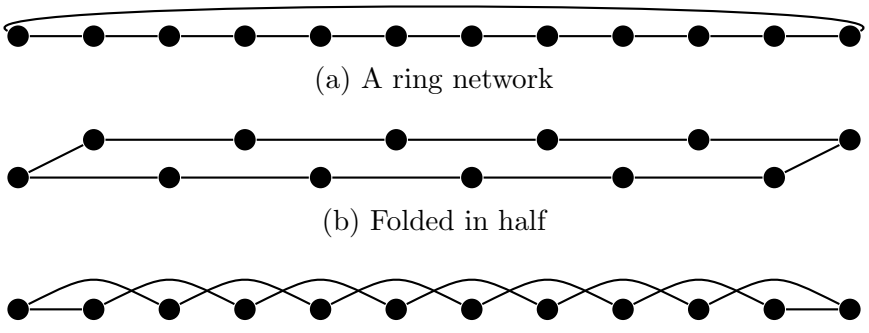


Figure 3.5: Logical arrangement of boards in a 4×4 threeboard SpiNNaker system. Touching boards are connected. Coloured lines represent long connections travelling North/South, North-East/South-West and East/West.

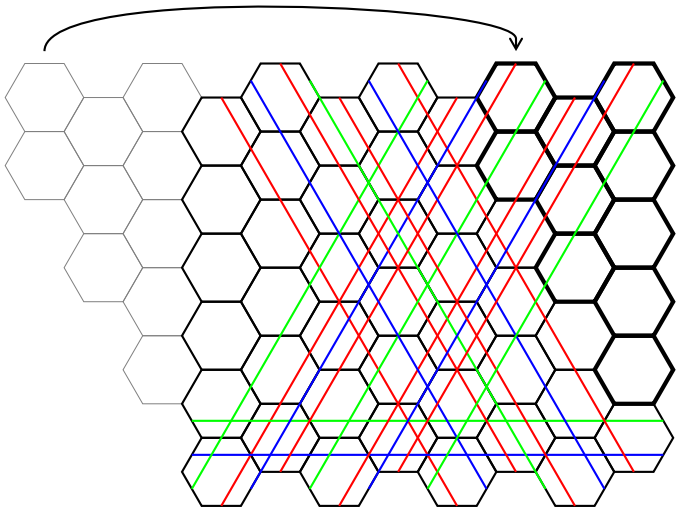


(a) A ring network

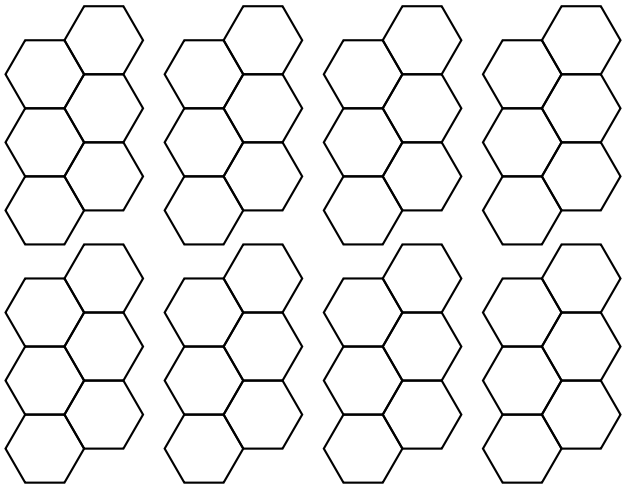
(b) Folded in half

(c) Interleaved

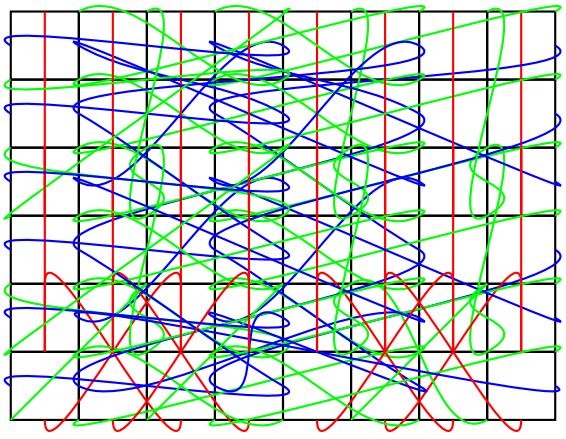
Figure 3.6: The process of folding a ring network to reduce the maximum wire length.



(a) Shift boards on the left to the right to form a rectangle.



(b) Fold along the gaps in this figure. (Wires omitted for clarity.)



(c) The (more complex) wiring after folding. (Shown here with squares as hexagons do not visually fit together after folding and interleaving.)

Figure 3.7: The process of folding SpiNNaker. Coloured lines represent wires travelling North/South, North-East/South-West and East/West.

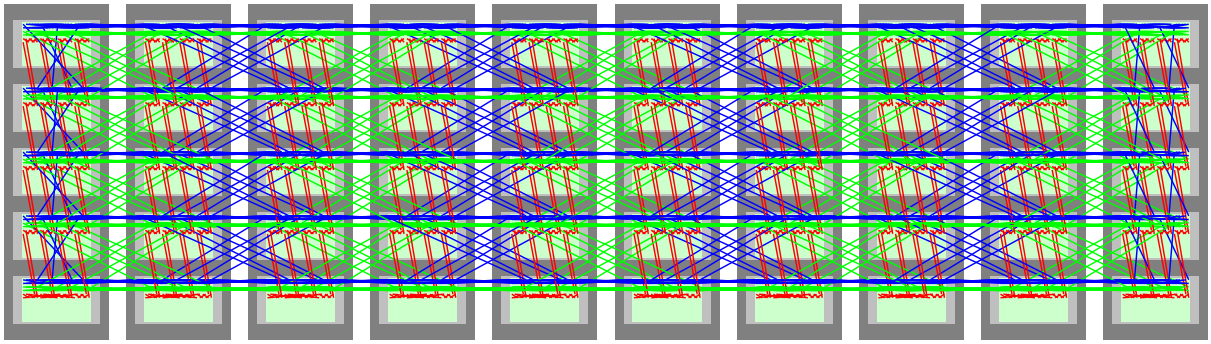


Figure 3.8: The largest SpiNNaker machine with 1,200 boards (20×20 threeboards) and 1,036,800 cores mapped into 10 cabinets of 5 racks each. Coloured lines represent wires connecting North/South, North-East/South-West and East/West links.

In the next step, the design is folded into four parts on the X-axis and into two on the Y-axis (figure 3.7b). It is necessary to fold the X-axis into four as the long, diagonal wires do not cross the entire system on the X-axis and instead reach half way. Folding in two would not bring these points any closer while folding into four brings them next to each other. For example, the wire travelling from the bottom left board to the top-middle board after folding in two would now have to cross from one end of the system to the other, an even longer distance than it had to before. As a result, the maximum wire length is reduced which can be seen in figure 3.7c. While the wiring in this image appears more complex, if only one direction (colour) is considered at once, a certain amount of regularity can be seen.

The final step in the process is to map the boards into their real-world physical positions. The largest SpiNNaker system will be installed into a series of cabinets, each containing a number of racks (shelves) into which the boards are slotted and wired up. Figure 3.8 shows a possible rack placement scheme for the largest planned SpiNNaker system of 20×20 threeboards.

Even though the system is physically several metres long, the longest wire will only be around one metre in length which is within the tolerances of the high-speed link technology. This placement scheme, therefore, meets the wire-length requirement, and due to the use of high-speed serial links using commodity cables as described earlier, the wire-cost requirement.

3.2.2 Wiring Complexity

The final concern is that it should not be too complex to be installed by hand. The large-scale SpiNNaker machine will contain 3,600 cables and so it is clearly not practical to have to look up each individual connection before wiring it up. Though it is clear that the folded arrangement is substantially less regular than the original layout, some regularity remains.

By splitting up the wiring tasks by the logical direction (and thus by the connector on the board) it is clear that a large amount of regularity exists in each group of cables (consider each colour separately in figure 3.8). If the task is further split depending on whether the cables stay within a rack or cabinet, batches of identical racks and cabinets

can be wired up independently before being linked together later. For example, the cables connecting North to South (shown in red) remain entirely within a cabinet meaning all North/South wiring can be completed independently for each cabinet.

Based on the above observations, the connections for all 3,600 cables can be described using only 53 instructions rather than 3,200. This satisfies the final constraint on wiring complexity.

3.2.3 'SpiNNer' Wiring Guide Generator

As part of this work SpiNNer, a wiring modelling library and wiring guide generator, was produced to assist in the construction of SpiNNaker systems built from multiple boards. These tools help avoid the error prone process of manually visualising the various transformations required during the mapping of boards to racks.

The modelling library allows transformations, such as folding, to be easily applied to a network of boards. From the transformed network, various measurements such as the maximum physical wire length can then be directly determined. Using this library the largest SpiNNaker machine can be described in 10 lines of code.

The wiring guide generator produces illustrated documents using LaTeX which describe the wiring for arbitrarily sized machines[1]. Metrics such as the distribution of wire lengths are included along with instructions for wiring the system up. At the time of writing, the largest prototype system constructed, a single threeboard, has been successfully assembled in a manner consistent with the generated wiring guide.

3.2.4 Further Work

The wiring guide generator is currently only capable of producing exhaustive wiring descriptions with one instruction for each of the 3,600 cables. The foundations for extracting regularity currently exist in experimental form and must eventually be integrated into the wiring guide generator.

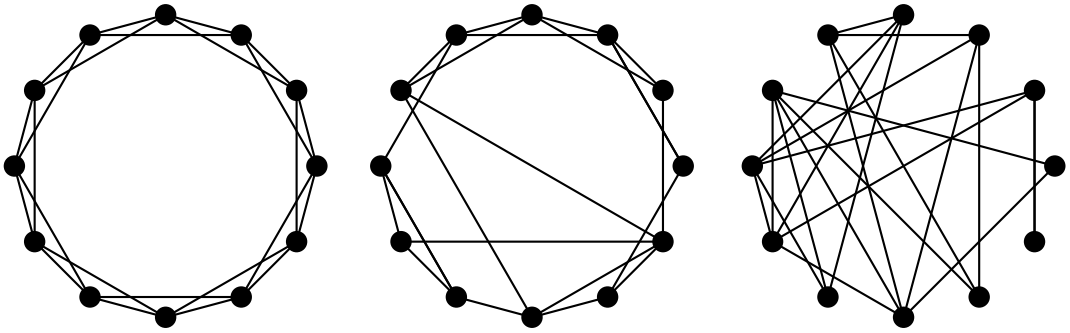
In addition, further work may be carried out to test alternative wiring schemes. In the current scheme, a large proportion of the cables cross between different cabinets. As a result a significant amount of the wiring cannot be done in isolated batches. An alternative scheme may be able to reduce this number and further simplify the task of wiring.

3.3 Small-World Super-Computers

Small-world networks are graphs with a very large number of nodes but, within which, the maximum shortest-path length is very small, despite each node having a relatively small number of edges. They also contain 'clusters' of well connected nodes, in contrast with random graphs (which may also fulfil the first criterion).

This type of network is often found in nature, perhaps most famously in social networks. This property was first observed in 1929 by Hungarian author Frigyes Karinthy[52]

[1]Many of the figures in this section were adapted from wiring guides generated by SpiNNer.



(a) Ring ($\beta = 0.0$) (b) Watts-Strogatz ($\beta = 0.2$) (c) Random ($\beta = 1.0$)

Figure 3.9: Watts-Strogatz networks with a range of rewiring probabilities.

and has become popularly known as the theory of 'six degrees of separation'. This theory states that for any two people, chosen at random, there is a chain of at most 6 acquaintances which connects them.

This property of maintaining low maximum shortest-path length while still remaining locally well connected is desirable for certain computational problems. In neural simulations most communication is local but with some longer connections existing. The clustering property of small world networks means these local connections can be well catered for while the low maximum shortest-path length means longer connections are still quick for very large models.

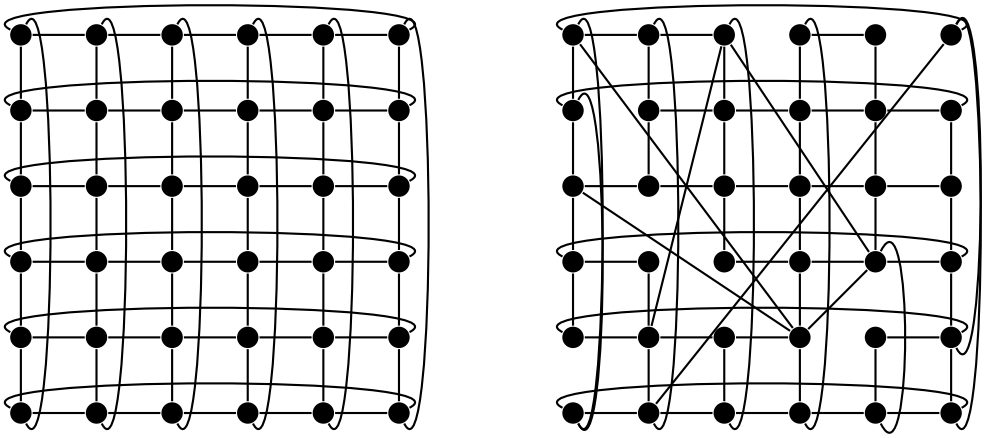
3.3.1 Network Construction

Watts and Strogatz have proposed an algorithm for randomly constructing networks with small-world properties [53]. The algorithm begins by creating a ring network with each node connecting to a fixed number of its neighbours (figure 3.9a). In the next step, with a probability of β, each edge may be replaced by a random connection. For $0 < \beta < 1$, the networks produced exhibit varying degrees of the small-world properties (figure 3.9b). Finally, in the extreme case where $\beta = 1$, the network devolves into a random network (figure 3.9c).

This algorithm is readily extended to torus topologies. In this work, a k-ary 2-cube (a two dimensional torus k nodes long in each dimension) was used as the initial network as in figure 3.10a. Random permutations are introduced as in the Watts-Strogatz model resulting in a network such as figure 3.10b.

A k-ary n-cube has an average shortest path length of $\frac{nk}{4}$. This is because a packet travels (on average, under uniform random traffic) $\frac{1}{4}$ of the way around each of the n, k-node-long dimensions. In this case, that means the average path length is $\frac{2\times6}{4} = 3$.

By contrast, for the rewired version shown, the average shortest path is now reduced to 2.77 hops. As in Watts-Strogatz networks, the average path length has been brought down while much of the local connectivity remains.



(a) Unmodified torus ($\beta = 0.0$) (b) Rewired torus ($\beta = 0.1$)

Figure 3.10: Extension of the Watts-Strogatz model to a 6-ary 2-cube.

3.3.2 Experiments

Experiments using a simple graph model were carried out to determine the effects of rewiring on shortest-path length. Figure 3.11 shows that the shortest-path length drops rapidly but, as more links are rewired, returns quickly diminish. As a result, it can be concluded that the amount of rewiring required to produce a significant impact on the shortest-path length can be very small. These results are similar to the findings of others such as Shin et al. who found an increase in bandwidth (but did not test latency) when using small-world networks [54].

Unfortunately, random wiring can be problematic for real-world systems. As discussed in §3.2, long wires must be avoided for many interconnection technologies and adding random wiring can cause many such links to be created. For example, in figure 3.10b a long wire from the top right node stretches almost to the bottom left node.

Work by others such as Koibuchi et al. have attempted to solve this problem in completely random topologies [55]. Unfortunately their approaches are not applicable to networks which already feature a high degree of structure such as the underlying torus network used in these experiments. As a result, a new scheme was devised.

To work around this problem, we first fold the network (as discussed previously in §3.2.1) to eliminate the long wrap-around wires which connect up the torus (figure 3.12a). Next, rewiring is carried out but with a limit on the maximum wire length which can be created (figure 3.12b). This rewiring constrains the maximum wire to be 3 units long (where a unit is the spacing between each node in a given dimension). Even with this limitation on wire length, the maximum path length still drops, this time to 2.73 (from 3 in the unwired network), but now there are no physically long wires.

To test the effects of limiting wire lengths when re-wiring, a parameter sweep was carried out for a 40-ary 2-cube. The test was carried out after folding once and twice in each dimension and finally when the network was not folded. The results are given in figure 3.13.

It is clear that when wire lengths are most limited, the average shortest-path length improves when the network has been folded. This result is intuitive because after folding,

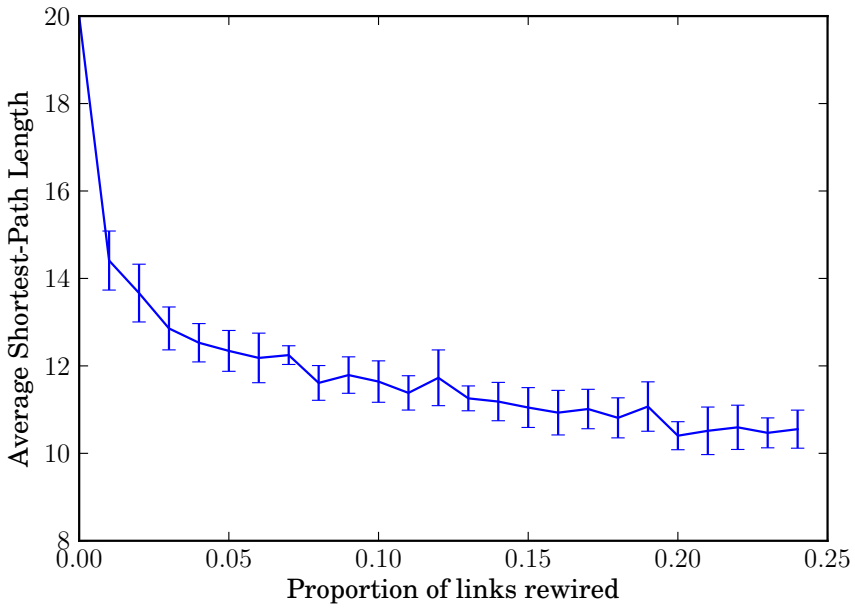
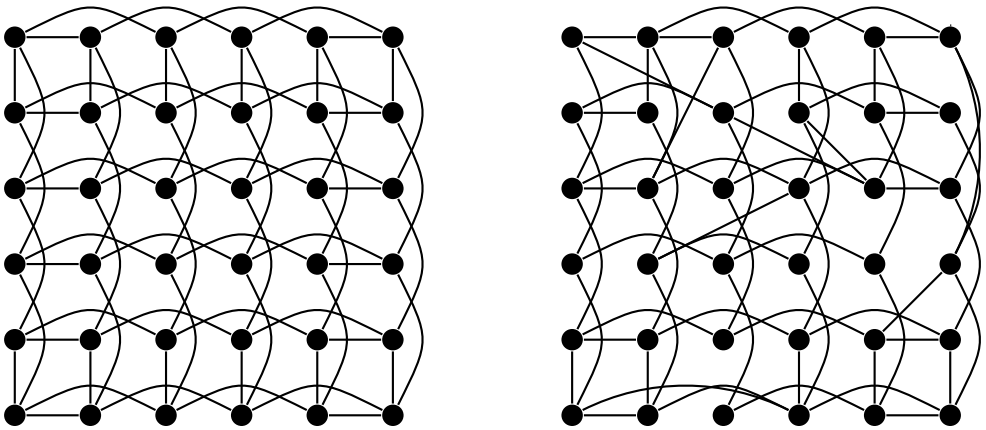


Figure 3.11: Average shortest-path length for folded 40-ary 2-cube (mean of 10 runs, error bars show 1 standard deviation).



(a) Folded torus ($\beta = 0.0$)

(b) Rewired torus ($\beta = 0.1$)

Figure 3.12: Rewiring a folded torus with constrained maximum wire-length.

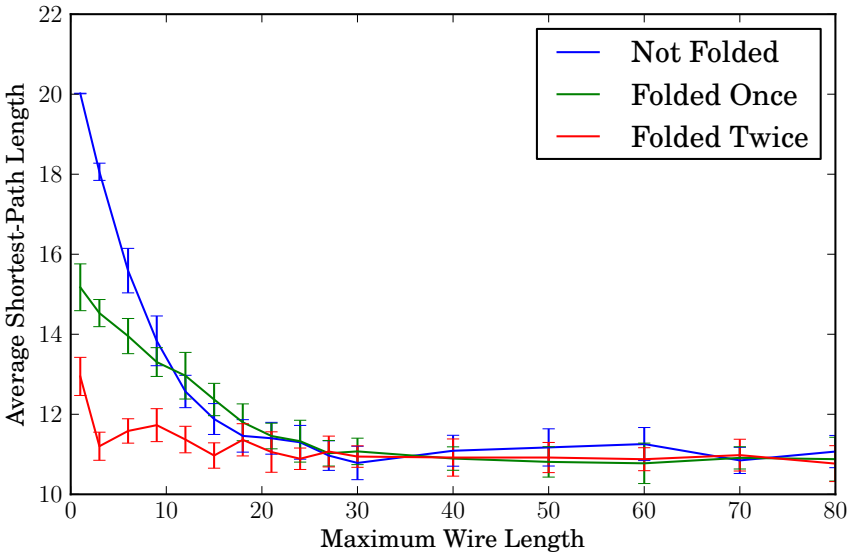


Figure 3.13: Average shortest-path length for folded 40-ary 2-cube with 1% rewiring

physically neighbouring boards are no-longer logically adjacent but instead are at opposite ends of the system. As a result, limiting wire lengths essentially 'prefers' the creation of logically longer links which are most likely to pull the average path length down.

The other interesting effect is that, above a certain threshold, allowing longer wires has no significant effect on the average shortest-path length. This may be because wires which stretch further than half-way ($\frac{k}{2}$ hops) along a given axis could be just as easily reached by travelling the shorter distance the other way around the axis. This longest useful wire length can be calculated as follows:

$$\text{Maximum Useful Wire Length} = \sqrt{n\left(\frac{k}{2}\right)^2} = \frac{k\sqrt{n}}{2}$$

For a 40-ary 2-cube the maximum useful wire length is $20\sqrt{2} \approx 28.3$ which can be visually confirmed in the figure.

3.3.3 Further Work

Figure 3.14 shows valid random links in a small ring network with limited wire lengths. Nodes near the top and bottom of the ring potentially have shorter average path lengths to all other nodes than nodes near the left and right of the ring. This is because the allowed random links for top and bottom nodes connect nodes at greater logical distances in the ring than those allowed on the left and right. The result is that the average path length from two different nodes becomes non-uniform across nodes in different parts of the system.

Further work may study the effects of this non-uniformity of average path length. The effects of the non-uniformity may be reduced by using greater degrees of folding or higher dimensional topologies. In addition, real systems are constrained to standard cabinets and racks, as in §3.2.1, which may change distribution of average path lengths in the system.

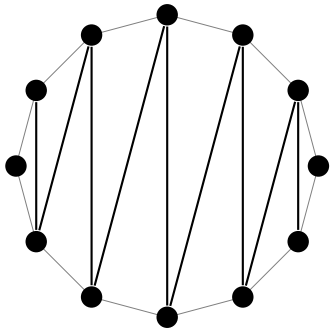


Figure 3.14: Valid random links in ring network when wires limited to a length of 1 unit and the network is folded as in figure 3.6.

This work could be extended to include this final mapping into more realistic physical placements, for example by building on the SpiNNer wiring modelling library. It is anticipated that the more complex placements resulting from higher-dimensions and realistic placement will result in a more uniform distribution of average path lengths.

Chapter 4

Research Plan

In this chapter an outline of the work to be carried out is presented. Due to the early stage of the research, planned later work is presented with less detail. Figure 4.1 shows a Gantt chart identifying the key stages of the project which are outlined in chronological order in the sections below.

4.1 SpiNNaker Modelling

As part of the ongoing SpiNNaker project a study is being carried out to compare how network simulations compare with the real hardware. As part of this work, researchers have built an FPGA-accelerated model of SpiNNaker's interconnection network. The work hopes to verify the accuracy of the model against both a traditional software model and the final silicon implementation. The results of this work are intended for a journal publication aiming for submission around the end of September 2013.

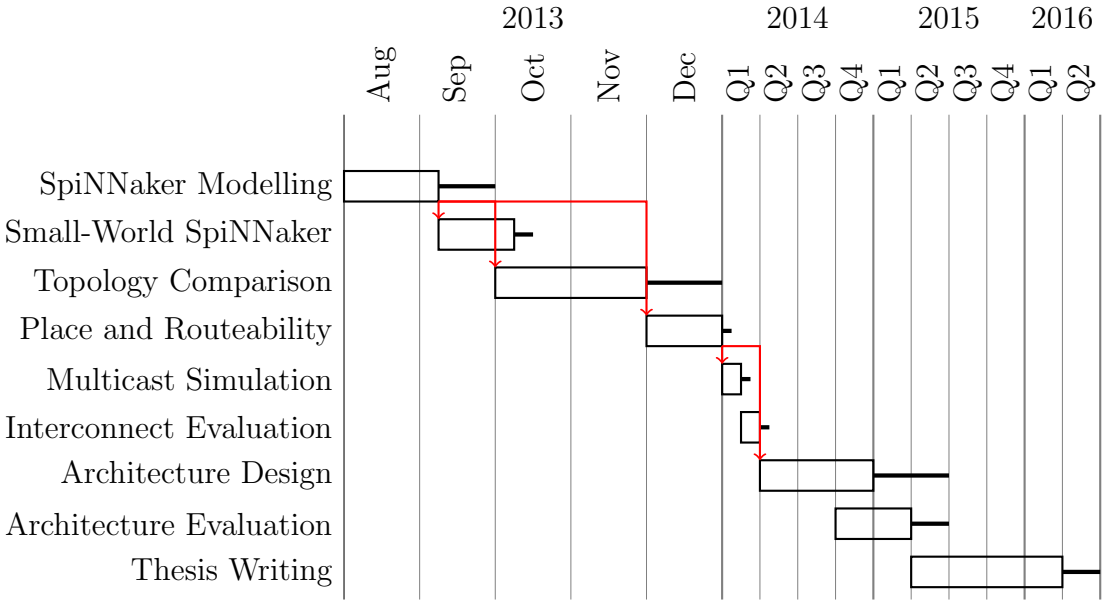
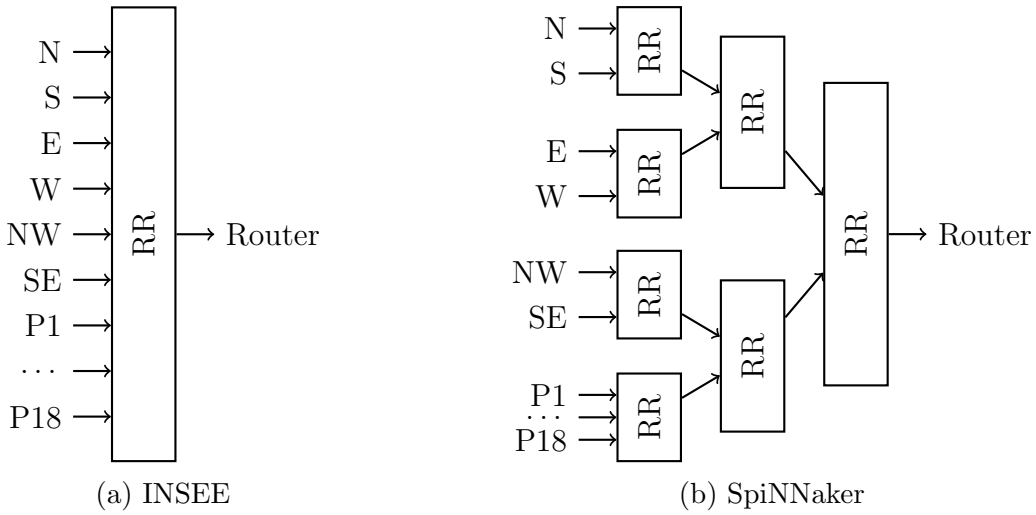


Figure 4.1: Gantt chart of proposed research plan. Arrows indicate dependencies, thick lines indicate slack. Note non-linear scale.



(a) INSEE (b) SpiNNaker

Figure 4.2: Incoming packet arbitration schemes in INSEE and SpiNNaker. Boxes marked 'RR' are round-robin arbiters, P1-P18 are connections to local processor cores.

My participation in this project will be to develop the software simulation to model and evaluate the network implemented in SpiNNaker. Once built, this simulator will be valuable for my own research into alternative network topologies. This section outlines in further detail the motivation and requirements for the simulator followed by a discussion of its applications for my own work.

4.1.1 Software Simulator Limitations

A prototype software model has been built using the INSEE simulator [56] which is designed to simulate a wide variety of networks. Unfortunately, INSEE's router model is different from that found in SpiNNaker and the FPGA model. The differences in their behaviour are outlined below.

One of the major limitations of INSEE is that it is only able to model point-to-point traffic and will require extending to support multicast traffic which is at the heart of SpiNNaker's anticipated usage for neural network simulation. However, since this limitation is also shared by the FPGA implementation, only point-to-point traffic is considered by this work in all three systems.

The other key difference between INSEE and the SpiNNaker architecture is the way packets from incoming links are arbitrated. Figure 4.2 shows the arbitration schemes for INSEE and SpiNNaker. INSEE uses a simple round-robin arbitration scheme while SpiNNaker (and the FPGA model) use a tree of arbiters[1].

The differences caused by this extend beyond the order in which contesting requests will be serviced. In INSEE, each input has a buffer associated with it from which the router will extract messages and forward them to the input buffer of the next node, one per simulated cycle. In the SpiNNaker design, messages are buffered at each level of the tree and eventually placed in a pipeline within the router (equivalent to further buffering)

[1]The bandwidth available at each level of the arbitration tree scales with the maximum input bandwidth for each level.

and an output buffer. The interaction of these buffers is not modelled by INSEE and thus the results produced will be less well matched with the actual SpiNNaker system.

4.1.2 Simulator Improvement Plan

Given the limitations of INSEE mentioned in the previous section, two possible approaches must be considered. Either INSEE must be modified to incorporate a more realistic model of the router or an alternative simulator must be used. One important factor in the decision is the 'ramp-up' time required to gain familiarity with the INSEE code-base compared with the time required to develop or extend an alternative simulator. The other factor is the utility of the simulator for my own research.

Since INSEE is an established tool which has been used in similar work it is potentially a strong choice. To determine its suitability for this work and my own research a small time will be initially spent analysing its design. If it is found to be suitable development of an extended version will commence.

A possible alternative to INSEE is the simulator developed during the preliminary interconnect study in §3.1. Like INSEE, it has already been used to simulate the SpiNNaker Interconnect and so configurations exist for SpiNNaker like machines. Because of the author's familiarity with the tool, extending the router model should be straightforward.

4.2 Small-World Network Experiments

Given the availability of an extensible and proven simulator, the next stage of my research will be to use it to model the behaviour of small-world style networks with more realistic traffic and wiring constraints.

The work done in §3.3 measured average shortest-path length in the networks examined. This measure corresponds to the average path length for uniform-random traffic in a real network with the same topology. In brain simulation systems such as SpiNNaker, much of the traffic is expected to be local and so uniform-random traffic is not representative. This work will test the performance of small-world networks with more realistic traffic.

The other shortcoming found during the preliminary study was that when physical wiring was restricted in length (as in real systems), the logical distance covered by links in different parts of the system becomes uneven. Since the folding and rack/cabinet placement which takes place in real machines such as SpiNNaker is much more intricate than the simple ring studied, these effects may become insignificant making wire-length limiting schemes profitable.

This work is intended to take around one further month including ample time for writing up and further experimentation with wiring schemes if required.

4.3 Topology Comparison

With the small-world network testing complete, the simulator should now be mature enough to begin further experiments with alternative, potentially more structured topolo-

gies. This will be done to empirically test alternative topologies for use in SNN simulation.

This work will follow on from the experiments carried out by Vainbrand and Ginosar which was limited to widely used, more generic network topologies with alternatives which may be better suited to the problem [5]. For example, topologies such as express cubes [57] offer improved performance for a limited amount of non-local traffic while preserving the local performance of mesh and torus networks.

4.4 Placement and Routing

Given the scale of the SpiNNaker machine and the networks it simulates, the NP-complete task of allocating processing tasks in each node in the system (placement) and routing messages between them (routing) is an important consideration. Current architectures, such as SpiNNaker, use table based routing where a lookup table is used to decide where to route messages arriving at each node in the system. Such systems offer a lot of flexibility in the routing schemes that can be implemented but may also introduce constraints on the complexity of routes through the system when the routing tables are limited in size.

The use of alternative topologies will require alternative routing schemes to be specified. In the above comparative work on possible topologies simple, naïve routing algorithms will be used. These simple to implement algorithms often result in non-optimal behaviour, especially with regard to load balancing [21]. Given that systems such as SpiNNaker use packet-dropping flow control, where congested routes can cause packets to be dropped, the effects of load imbalance may cause localised areas of high packet losses which may significantly affect simulations [58].

In this phase of work alternative routing algorithms which attempt to handle routing table depth and congestion control will be studied. The former problem is similar in style to the problem faced by chip and printed circuit design automation tools. The latter is largely covered by conventional interconnect/network routing algorithms. Combining and evaluating these two approaches will be where much of this time is spent.

4.5 Effects of Multicast

Following on from work on basic routing, the challenge of performing routing for multicast traffic must be considered. Multicast routing must route a message from a single source to multiple destinations, in the case of spiking neural networks, potentially thousands of destinations.

Current approaches to multicast routing again are split between chip and circuit design and interconnect/network routing. Chip routers often constrain the paths in various ways, for example to ensure that latency to each destination is similar. This type of constraint is similar in style to those in SpiNNaker-like systems for example, the need to try and control branching to minimise routing table resource. Network routers, however, are often designed to reduce the bandwidth consumed by forking multipath routes as late as possible reducing the total number of links used. Once again, work must be done to combine and compare these two approaches.

4.6 Interconnect Technology Evaluation

Given a suitable topology, a further question is that of the actual implementation of the interconnect. Historically small-scale systems have used synchronous signals to communicate between various parts of the system. As systems have scaled, this is no longer feasible as clock distribution grows ever more difficult. Instead architectures have become prominent where individual, locally-synchronous components communicate via a locally-synchronous interconnect.

Internally, the SpiNNaker chip uses an asynchronous NoC based on CHAIN [59, 60] using IP from (the now defunct) Silistix Ltd. This interconnect is able to handle arbitration between the many incoming signals from on-board processors and external connections to other chips. Since this technology is no longer available, alternatives must be sought.

SpiNNaker chips are interconnected using parallel, delay insensitive interconnects. These links, while adequate for the system's current scale, may not be trivially speeded up. More suitable external interconnects will likely be based on high speed serial as described in §2.5.

4.7 'SpiNNaker 2' Architecture

Based on the preceding work and on further developments within the SpiNNaker project, a new architecture for neural network simulation will be proposed. In particular the architecture is intended to be a successor to the SpiNNaker platform aimed at a similar communication-bound model.

4.7.1 Scaling

Due to the advance of Moore's law, several times more logical capacity will be available for this next generation architecture compared to the previous revision. It is assumed that it will be possible to fit a larger number of cores into a single chip along with larger memories both for the on-board processors and routers.

The preceding work will be used to decide on the type of topology and routing systems to be used. It is expected that access to larger routing table resource as well as more area for routing logic will be available.

4.7.2 Interconnection Network

One of the known limitations of SpiNNaker's interconnection network is that it has been optimised largely for the style of traffic required while simulations are in progress, in particular it deals very well with small, multicast packets whose delivery does not need to be guaranteed, as with biological spikes.

Other important tasks, however, are not handled as well such as initialisation and result collection. These tasks require reliable communication, typically with a relatively larger data payload. This is currently facilitated by the SpiNNaker Datagram Protocol (SDP) [61] which allows non-guaranteed point-to-point transmission of up to 64 kilobytes of data. Though this offers the ability to load configurations to SpiNNaker chips, collecting data becomes more difficult. With larger, multi-board systems currently in testing,

contention for the links surrounding a central node during data collection become saturated resulting in a large number of packets being dropped.

4.7.3 Link Technology

The underlying Silistix technology used by SpiNNaker's links is not available for new systems and must be replaced. Based on research into link technologies a decision must be made for the new system. This choice will strongly influence the design of the network.

4.7.4 Evaluation

The proposed topology will be evaluated by simulation building on the simulators developed earlier in the project. It is hoped that more information about the actual usage and traffic patterns encountered by the current SpiNNaker hardware will be available to drive this simulation and to present a point of comparison.

Chapter 5

Conclusion

The challenge of understanding the brain has led to the construction of vast simulated neural models. Over time, models have increased in size, complexity and realism. Though the Blue Brain project has simulated detailed models of small systems of hundreds of thousands of neurons using an 'off-the-shelf' super-computer, such models pale in scale to the human brain with its 100 billion neurons. Large networks of simple neurons, such as Spaun, have shown that complex and biologically realistic behaviour can be generated by simple models of neural behaviour.

Conventional super-computers feature large amounts of computational resource interconnected by a topology designed for moving large blocks of data at high bandwidths. Large scale simulations of simple neuron models do not fit the expectations of such machines as they feature only a limited computational need instead requiring the transmission of large numbers of extremely small or empty messages with a high sensitivity to latency. This mismatch has led to the development of special-purpose architectures for brain simulation.

Many brain simulation architectures have been built on custom analogue and digital circuits which support certain models of neuron. Such devices have achieved extremely efficient simulations of large numbers of neurons, for example the Neurogrid architecture is able to simulate one million neurons while consuming orders of magnitude less power than the super-computer used for Blue Brain. Meanwhile the BrainScaleS project has produced a wafer-scale system which can simulate neural models up to tens of thousands of times faster than their biological counterparts.

Though typically fast and power efficient, the lack the flexibility to run other neural models than those intended for them. As well as being computationally limited, their topologies also often place constraints on their scalability. For example, the Neurogrid topology fails to exploit the on-chip parallelism available in their architecture allowing only one neuron to spike at a given time. Other systems, such as BrainScaleS, feature topologies whose dimensions are limited to that which can be manufactured in a single silicon wafer or chip.

These limitations are being tackled by the SpiNNaker system which combines large numbers of small-low power general purpose processors in a scalable network with the aim of simulating networks of one billion neurons in biological real time. The SpiNNaker topology forms a regular 2D toroid built out of boards containing 48 chips connected together by delay-insensitive asynchronous parallel links. The boards are then connected

using high-speed serial links to reduce the number of wires required to connect the boards together.

A simulation of the non-homogeneous topology used by SpiNNaker was analysed to test the effects on the network latency. Though an 80% increase in latency was measured, this is not expected to significantly affect the system's real-time performance. Since the expected median latency increases to only 7.5 μs this is still far shorter than the 1 ms time steps used by planned neural simulations. Should the period of the time-steps be reduced to improve simulation accuracy this problem may become a problem.

A preliminary model of an unconventional semi-random topology was assessed as an alternative with results showing improved latencies. Unlike previous work, the issue of wiring practicality is addressed and results suggest that gains can still be made despite restrictions on wire placement.

The issue of wiring practicality was also investigated for SpiNNaker's existing topology. A wiring scheme was developed which ensures that cables remain short enough for the high-speed serial communication scheme to function when constrained to board placements in standard computer cabinets. In addition, the 3,600 wiring instructions was reduced to just 53 making construction of large SpiNNaker machines practical.

Work on a new interconnect simulator will allow for the analysis of alternative topologies, including the semi-random topology already developed, to be tested in a more realistic manner. This simulator will be compared against actual SpiNNaker hardware and another, hardware-accelerated simulator which will allow the accuracy of the simulator to be assessed.

Based on further refinement of the simulation to account for details such as multicast traffic and the issue of resource placement and routing, a new architecture will be developed to facilitate the next generation of neural simulations. With the continued advance of Moore's law, such an architecture may be able to approach models whose size is within an order of magnitude of the human brain.

Bibliography

[1] IBM. Pin-point cancer therapy: Targeting only bad cells with proton radiation. `http://www.research.ibm.com/articles/proton-radiation-therapy.shtml`. Accessed 27 June 2013.

[2] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Charlie Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, 2012.

[3] Henry Markram. The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[4] SB Furber, Steve Temple, and AD Brown. High-performance computing for systems of spiking neurons. In *AISB06 workshop on GC5: Architecture of Brain and Mind*, volume 2, pages 29–36, 2006.

[5] Dmitri Vainbrand and Ran Ginosar. Scalable network-on-chip architecture for configurable neural networks. *Microprocessors and Microsystems*, 35(2):152–166, 2011.

[6] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[7] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[8] Wolfgang Maass and Christopher M Bishop. *Pulsed neural networks*. MIT press, 2001.

[9] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in human neuroscience*, 3, 2009.

[10] Jack Dongarra. Visit to the National University for Defense Technology Changsha, China. `http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf`, June 2013. Accessed: 17 July 2013.

[11] Arthur S Bland, Jack Wells, Otis E Messer, Oscar Hernandez, and Jim Rogers. Titan: Early experience with the Cray XK6 at Oak Ridge National Laboratory. *Cray User Group*, 2012.

[12] Timothy Prickett Morgan. IBM uncloaks 20 petaflops BlueGene/Q super computer. `http://www.theregister.co.uk/2010/11/22/ibm_blue_gene_q_super/`, November 2010. Accessed: 17 July 2013.

[13] Fujitsu Limited. Supercomputer "K computer" Takes First Place in World. http://www.fujitsu.com/global/news/pr/archives/month/2011/20110620-02.html, June 2011. Accessed: 17 July 2013.

[14] M. Yokokawa, F. Shoji, A. Uno, M. Kurokawa, and T. Watanabe. The K computer: Japanese next-generation supercomputer development project. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 371–372, 2011.

[15] Hans Meuer. Top500 List - June 2013. http://www.top500.org/list/2013/06/, June 2013. Accessed: 17 July 2013.

[16] Jack Dongarra, Jim Bunch, Cleve Moler, and Gilbert Stewart. LINPACK Benchmark Homepage. http://www.netlib.org/linpack/. Accessed: 17 July 2013.

[17] Jack Dongarra. LINPACK Readme. http://www.netlib.org/linpack/readme, Febuary 1984. Accessed: 19 July 2013.

[18] Jack Dongarra. The HPC Challenge Benchmark: A Candidate for Replacing Linpack in the Top500? Keynote, 2007 SPEC Benchmark Workshop, January 2007.

[19] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. Introducing the graph 500. *Cray Users Group (CUG)*, 2010.

[20] Richard Murphy. Graph500 List - June 2013. http://www.graph500.org/results_jun_2013, June 2013. Accessed: 19 July 2013.

[21] William James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004.

[22] Vitali Morozov, Jiayuan Meng, Venkatram Vishwanath, Jeff R Hammond, Kalyan Kumaran, and Michael E Papka. ALCF MPI Benchmarks: Understanding Machine-Specific Communication Behavior. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 19–28. IEEE, 2012.

[23] Dong Chen, Noel A Eisley, Philip Heidelberger, Robert M Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L Satterfield, Burkhard Steinmacher-Burow, and Jeffrey J Parker. The IBM Blue Gene/Q interconnection network and message unit. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–10. IEEE, 2011.

[24] HP Graf, LD Jackel, RE Howard, B Straughn, JS Denker, W Hubbard, DM Tennant, and D Schwartz. VLSI implementation of a neural network memory with several hundreds of neurons. In *AIP conference proceedings*, volume 151, page 182, 1986.

[25] Mark Holler, Simon Tam, Hernan Castro, and Ronald Benson. An electrically trainable artificial neural network (etann) with 10240 'floating gate' synapses. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 191–196. IEEE, 1989.

[26] Aharon J Agranat, Charles F Neugebauer, and Amnon Yariv. A CCD based neural network integrated circuit with 64K analog programmable synapses. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 551–555. IEEE, 1990.

[27] Mostafa Rahimi Azghadi, Said Al-Sarawi, Derek Abbott, and Nicolangelo Iannella. A neuromorphic VLSI design for spike timing and rate based synaptic plasticity. *Neural Networks*, 2013.

[28] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(13):239 – 255, 2010.

[29] Guy Rachmuth, Harel Z. Shouval, Mark F. Bear, and Chi-Sang Poon. A biophysically-based neuromorphic model of spike rate- and timing-dependent plasticity. *Proceedings of the National Academy of Sciences*, 2011.

[30] Swadesh Choudhary, Steven Sloan, Sam Fok, Alexander Neckar, Eric Trautmann, Peiran Gao, Terry Stewart, Chris Eliasmith, and Kwabena Boahen. Silicon neurons that compute. In *Artificial Neural Networks and Machine Learning–ICANN 2012*, pages 121–128. Springer, 2012.

[31] Liam P Maguire, T Martin McGinnity, Brendan Glackin, Arfan Ghani, Ammar Belatreche, and Jim Harkin. Challenges for large-scale implementations of spiking neural networks on fpgas. *Neurocomputing*, 71(1):13–29, 2007.

[32] SJ Prange and H Klar. Cascadable digital emulator IC for 16 biological neurons. In *Solid-State Circuits Conference, 1993. Digest of Technical Papers. 40th ISSCC., 1993 IEEE International*, pages 234–235. IEEE, 1993.

[33] Axel Jahnke, Ulrich Roth, and Heinrich Klar. A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN). In *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on*, pages 232–237. IEEE, 1996.

[34] T Schoenauer, N Mehrtash, Andreas Jahnke, and H Klar. Maspinn: Novel concepts for a neuroaccelerator for spiking neural networks. In *Ninth Workshop on Virtual Intelligence/Dynamic Neural Networks: Neural Networks Fuzzy Systems, Evolutionary Systems and Virtual Re*, pages 87–96. International Society for Optics and Photonics, 1999.

[35] Nasser Mehrtash, Dietmar Jung, Heik Heinrich Hellmich, Tim Schoenauer, Vi Thanh Lu, and Heinrich Klar. Synaptic plasticity in spiking neural networks (SP^2INN): a system approach. *Neural Networks, IEEE Transactions on*, 14(5):980–992, 2003.

[36] HH Hellmich, M Geike, P Griep, P Mahr, M Rafanelli, and H Klar. Emulation engine for spiking neurons and adaptive synaptic weights. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 5, pages 3261–3266. IEEE, 2005.

[37] Steve Furber and Steve Temple. Neural systems engineering. *Journal of the Royal Society interface*, 4(13):193–206, 2007.

[38] Kwabena A Boahen. A burst-mode word-serial address-event link-i: Transmitter design. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(7):1269–1280, 2004.

[39] Kwabena A Boahen. A burst-mode word-serial address-event link-ii: Receiver design. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(7):1281–1291, 2004.

[40] Johannes Schemmel, Johannes Fieres, and Karlheinz Meier. Wafer-scale integration of analog neural networks. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 431–438. IEEE, 2008.

[41] Johannes Schemmel, D Bruderle, A Grubl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 1947–1950. IEEE, 2010.

[42] Johannes Fieres, Johannes Schemmel, and Karlheinz Meier. Realizing biological spiking network models in a configurable wafer-scale hardware system. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 969–976. IEEE, 2008.

[43] Arjun Singh, William J Dally, Brian Towles, and Amit K Gupta. Locality-preserving randomized oblivious routing on torus networks. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 9–13. ACM, 2002.

[44] F. Garcia Nocetti, I. Stojmenovic, and J. Zhang. Addressing and routing in hexagonal networks with applications for tracking mobile users and connection rerouting in cellular networks. *Parallel and Distributed Systems, IEEE Transactions on*, 13(9):963–971, 2002.

[45] Simon Davidson. Spinnaker routing algorithm: Constraints and some proposed solutions. An internal document proposing multicast routing algorithms., July 2013.

[46] Sergio Davies, Cameron Patterson, Francesco Galluppi, Alexander Rast, David Lester, and Steve Furber. Interfacing real-time spiking I/O with the SpiNNaker neuromimetic architecture. *Submitted to ICONIP*, 2010.

[47] SATA-IO. SATA-IO Homepage. `https://www.sata-io.org/`. Accessed: 21 July 2013.

[48] InfiniBand Trade Association. InfiniBand Trade Association Homepage. `http://www.infinibandta.org/`. Accessed: 21 July 2013.

[49] J. Navaridas, M. Luján, J. Miguel-Alonso, L.A. Plana, and S. Furber. Understanding the interconnection network of SpiNNaker. In *Proceedings of the 23rd international conference on Supercomputing*, pages 286–295. ACM, 2009.

[50] Simon Davidson. Summary of building a toroid using hexagons, v1.2. An internal document describing how how chips are distributed between boards and how boards can be linked to form a toroid.

[51] Steve Furber. Wiring boards up. Correspondence by Email, April 2013. A discussion on methods of arranging boards to minimise wire length.

[52] Frigyes Karinthy. *Chain-Links*. Public Domain, 1929. Translated from Hungarian and annotated by Adam Makkai and Enikö Jankó.

[53] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[54] Ji-Yong Shin, Bernard Wong, and Emin Gün Sirer. Small-world datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 2:1–2:13. ACM, 2011.

[55] Michihiro Koibuchi, Ikki Fujiwara, Hiroki Matsutani, and Henri Casanova. Layout-conscious random topologies for hpc off-chip interconnects. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 484–495, 2013.

[56] Javier Navaridas, Jose Miguel-Alonso, Jose A Pascual, and Francisco J Ridruejo. Simulating and evaluating interconnection networks with INSEE. *Simulation Modelling Practice and Theory*, 19(1):494–515, 2011.

[57] William J. Dally. Express cubes: improving the performance of k-ary n-cube interconnection networks. *Computers, IEEE Transactions on*, 40(9):1016–1023, 1991.

[58] Daniel L. Greenfield. *Communication Locality in Computation, Software, Chip Multiprocessors and Brains*. PhD thesis, University of Cambridge Computer Laboratory, April 2010.

[59] Luis A Plana, Steve B Furber, Steve Temple, Mukaram Khan, Yebin Shi, Jian Wu, and Shufan Yang. A GALS infrastructure for a massively parallel multiprocessor. *Design & Test of Computers, IEEE*, 24(5):454–463, 2007.

[60] John Bainbridge and Steve Furber. Chain: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, 2002.

[61] Steve Temple. *AppNote 4 - SpiNNaker Datagram Protocol (SDP) Specification*. University of Manchester: Advanced Processor Technologies Group, November 2011.