Studying the Effects on Packet Latency of Naive Routing in SpiNNaker's Heterogeneous Interconnection Network

Jonathan Heathcote

Abstract

The SpiNNaker project aims to produce a massively parallel machine containing one million low-power processor cores for running neural simulations in real-time. The system will consist of multiple circuit boards each hosting 864 ARM processors on 48 chips connected in a hexagonal mesh topology. In order to connect the boards together, the outer edges of the boards will be joined together to form a large toroidal mesh via a number of S-ATA links. A simulator was built to asses the effects of these links on packet latency when naive dimension-order routing is used. Our results show that a latency overhead of 80% is introduced by the links. The 'emergency routing' protocol implemented by SpiNNaker was also observed to interact non-optimally with the S-ATA links. Possible measures are proposed which may alleviate some of the problems observed.

1 Introduction

The SpiNNaker system is a machine designed to emulating the behaviour of the brain using a novel, massively-parallel system architecture. The system will ultimately be made up of 1,036,800 low-power ARM processors simulating a network of one billion neuron models in real-time. As such, it is important that messages can travel around the system with relatively low latencies.

A machine will be built out of circuit boards containing 48 18-core SpiNNaker chips. The chips are connected to each-other by delay-insensitive links consisting of 8 wires per link. Linking the boards together using this type of link is not practical: hundreds of inter-board wires would be required coming with a significant performance and power cost. Instead, these links will be transparently multiplexed onto six high-speed S-ATA based links with equivalent total bandwidth.

A high-level simulator was built to observe the interaction between the naive dimension-order routing algorithm used by current SpiNNaker systems and the proposed heterogeneous network.

In section 2 the SpiNNaker architecture and topology is introduced. In section 3 the design of our simulator is described along with the modelling simplifications made. Finally, experimental results from our simulations are presented in 4 followed by conclusions and proposed future work in 5.

2 SpiNNaker Architecture

This section introduces the topology of the interconnection network used in the SpiNNaker system followed by the proposed inter-board S-ATA links.

2.1 Topology

The smallest unit in the system is the SpiNNaker chip consisting of 18 low-power ARM processors, a router and some memory [6]. Each chip is connected to six of its neighbours to the as shown in figure 1. These links use a pair of delay-insensitive 8-wire 2-of-7 non-return-to-zero (NRZ) connections to transfer packets of data.

The system is made up of a grid of chips in which the outermost connections wrap around into the shape of a torus. This transformation can be visualised by imagining the top half of the mesh rolled around to meet the bottom half

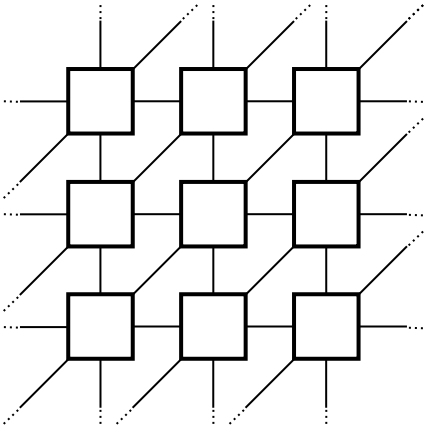


Figure 1: Chips in the SpiNNaker system are connected to their north, north-east, east, south, south-west and west neighbours. Connections on the extreme edges of the system are connected together forming a torus (not shown here).

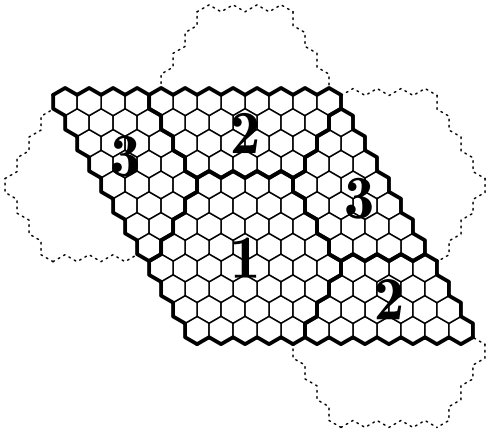


Figure 3: An minimal arrangement of three boards which forms a torus when the opposing edges are attached.

forming a tube. This tube is then bent around so that the two ends meet forming a torus.

Boards containing forty-eight SpiNNaker chips are the next largest unit in the system. The chips are arranged into a hexagon as shown in figure 2. This exposes six pairs of complementary edges of equal numbers of links which can be connected to other boards.

Three boards can be combined as shown in figure 3 to form a torus. This formation can be repeated in order to produce larger systems. The final, full-scale SpiNNaker machine will feature 20 repeats of this formation horizontally and vertically to produce a system with 57,600 chips (containing 1,036,800 ARM cores).

2.2 Serial-ATA Links

The six sets of eight edge links are multiplexed onto six high-speed Serial-ATA links to connect boards together. These links are driven by the three on board FPGAs (each driving two of the links labelled in figure 2).

Packets from the incoming links are assembled into frames by the FPGAs and then sent via the S-ATA link to the another board where they are disassembled into SpiNNaker packets and demultiplexed to the associated chips [4].

The S-ATA links have a theoretical maximum bandwidth of 3.2 gigabits/s [8] which is greater than the total $8 \times 0.25 = 2.0$ gigabits/s bandwidth of the multiplexed links. This means that bandwidth will not be reduced across the multiplexed links however the latency will be increased.

3 Simulator

In order to observe the effects of the multiplexed links, a simplified model of the SpiNNaker system was simulated. This model aims to be an extension of the simulation used by Navaridas et al. [5] extended to include these new links.

Due to the size and complexity of the full SpiNNaker system, this simulation uses a relatively crude model similar to [5]. This section presents the simplifications made and describes the choice of simulation system used.

3.1 Modelling Simplifications

The system is reduced to a network of nodes containing a router and traffic generator (figure 4) modelling a single SpiNNaker chip. All 18 on-board ARM cores and their software are modelled by the node's single random traffic generator.

These nodes are arranged into boards, as in the real SpiNNaker system, with models of

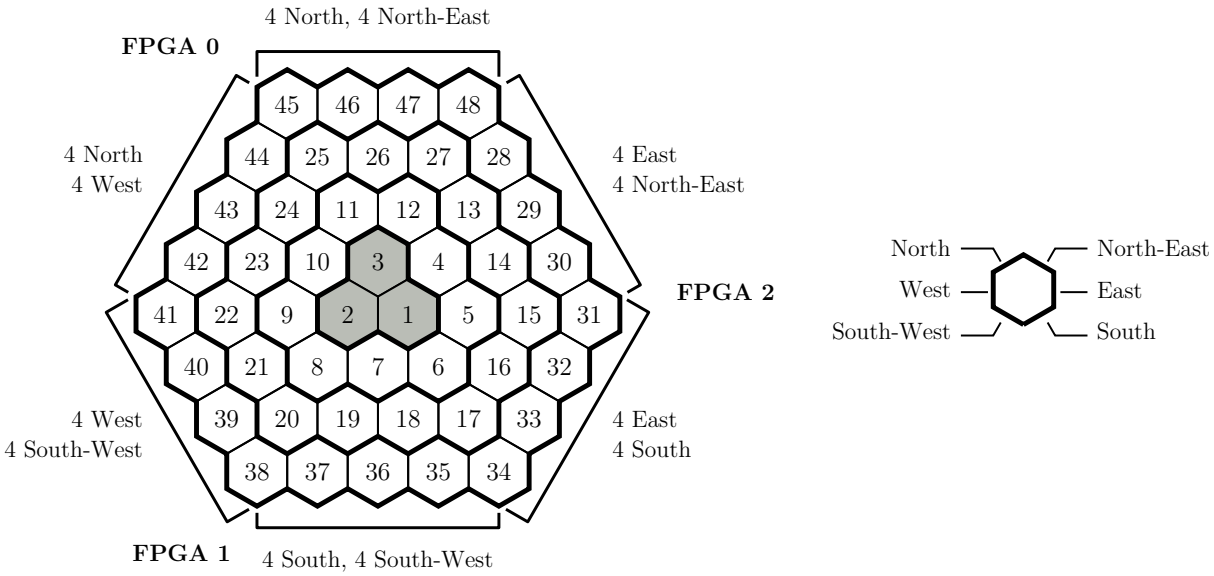


Figure 2: The logical arrangement of SpiNNaker chips on a board. Each hexagon represents one SpiNNaker chip. Chips which share an edge are linked. The external links are divided up into six groups as shown with opposing edges featuring complementary links. The edge links are connected to three on-board FPGAs used to multiplex the links onto high-speed S-ATA connections. The hexagonal pattern is formed layer-by-layer around the 3-hexagon kernel shown in grey. The boundaries between subsequent layers of the hexagon's structure are shown in bold.

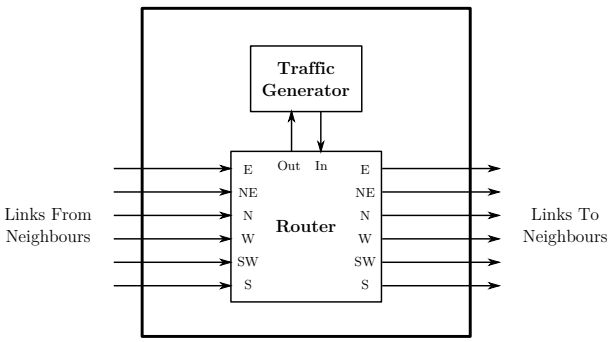


Figure 4: A node in the simulator representing one SpiNNaker chip. The traffic generator models all 18 ARM cores.

2-of-7 links connecting nodes to their neighbours. Multiple boards are finally arranged into a torus and connected together by models of the proposed S-ATA link.

The simulation is clocked at 150MHz: the clock speed of the processor, router and S-ATA link hardware in current real hardware configurations. As in previous work, the discretisation of the simulation is not considered to have a major effect on model accuracy.

3.1.1 Traffic Generation

All traffic in the simulated network is point-to-point (P2P) ignoring SpiNNaker's multicast routing facilities but greatly simplifying the task of generating random packets as well as simplifying the task of routing.

Traffic is generated by a Bernoulli process where, at a rate of once per cycle, an random decision is made whether to transmit a new packet. Packets are sent to a random destination selected from a uniform distribution. As in [5], a packet is generated by each traffic generator every cycle with a probability of 0.01.

Generated packets are placed into a four-packet buffer before being routed. If the buffer is full then new packets are dropped until buffer space is available again.

Packets are generated with a fixed size of 40 bits corresponding to a 'short' SpiNNaker packet. 'Long' packets of 72 bits are not included in the simulation.

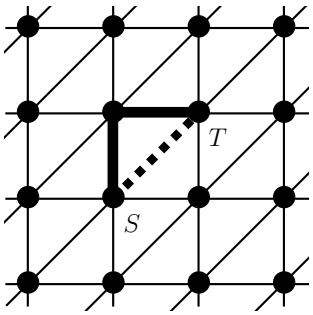


Figure 5: When sending a packet from node S to T, an emergency route (thick line) is used if intended link (dashed) is blocked for too long.

3.1.2 Routing

The router modelled in the simulation uses simple dimension order routing as used by current SpiNNaker routing software for P2P packets. As in the previous model, the router is able to route one packet from every input when the corresponding output is ready every cycle. A round-robin scheme is used for prioritising the inputs' access to output ports.

After 240[1] cycles, unrouted packets enter emergency routing mode where they are routed via the two sides of a right-angled triangle one link counter-clockwise (figure 5). If the message cannot be sent after a further 240 cycles, the packet is dropped.

The coordinate system described in [3] defines the dimensions as shown in figure 6a. Also defined are a corresponding set of unit vectors $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$.

All nodes are given coordinates relative to the bottom-left node (labelled A in the figure) defined to be at $(0, 0, 0)$. Coordinates in this system are not unique due to the non-orthogonal axes. For example, travelling along the vector $\boldsymbol{i}+\boldsymbol{j}+\boldsymbol{k}$ from $(0, 0, 0)$ reaches $(1, 1, 1)$ which is at exactly the same position. This clearly gives each point an infinite set of valid coordinates.

Definition 1. *As described in [3], a vector representing a shortest path between two points is of the form $a\boldsymbol{i}+b\boldsymbol{j}+c\boldsymbol{k}$ with the following constraints:*

- *$abc = 0$, that is, at most two dimensions are used (and non-zero) and at least one isn't (and is zero).*
- *The non-zero dimensions have opposite signs.*

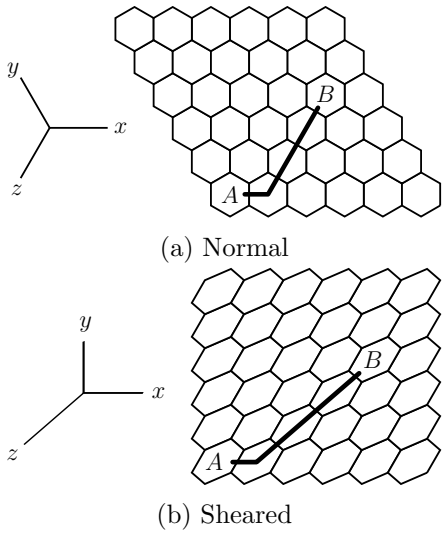
For example, nodes A and B in the figure are at positions which can be written as $(0, 0, 0)$ and $(4, 3, 0)$ respectively. These particular forms of coordinate are especially intuitive when the diagram is sheared to line the x and y axes up with their conventional counterparts as in figure 6b. A vector between these points can be found by subtracting the terms yielding $4\boldsymbol{i} + 3\boldsymbol{j} + 0\boldsymbol{k}$. This vector, however, does not represent the shortest path as the two non-zero components have the same sign violating definition 1.

As shown previously, the vector $\boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k}$ represents a path which leads back to where it started. As a result we can add or subtract multiples of this vector to yield a vector matching definition 1. In this case, $4\boldsymbol{i} + 3\boldsymbol{j} + 0\boldsymbol{k} - 3(\boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k}) = 1\boldsymbol{i} + 0\boldsymbol{j} + -3\boldsymbol{k}$ or simply $\boldsymbol{i} - 3\boldsymbol{k}$ which is the shortest path vector from A to B. Dimension order routing is then applied to yield the path shown.

This process can be extended to apply to a toroidal mesh where the edges of the mesh wrap-around. The starting and ending points are translated into three forms such that the starting point is at $(0, 0, 0)$ followed by $(\frac{w}{2}, \frac{h}{2}, 0)$ and $(w, h, 0)$ with w and h being the width and height of the mesh respectively. The coordinates wrap around the mesh boundaries accordingly. The shortest path vector for each of these forms is then found and the shortest selected[2].

[1] The number of cycles before emergency routing takes place is configurable but this value is used as the default selected by the current SpiNNaker configuration tools.

[2] This method is exceedingly inelegant and was selected after several days of working on this problem. It is hoped that a more elegant solution exists and finding it is left as an exercise for the more geometry-aware reader.



(a) Normal

(b) Sheared

Figure 6: An example of a shortest-path between A and B on a hexagonal mesh. Shown both normally and sheared to yield the familiar 2D-mesh style layout. Note that the Z axis elongated which can make interpreting distances unintuitive in this projection.

3.1.3 Links

The 2-of-7 delay insensitive links are modelled as taking 23 cycles for a complete 40 bit packet to arrive and a further 1 cycle for the final acknowledgement to arrive back at the sender allowing the next packet to arrive.

The S-ATA links are modelled by the system depicted in figure 7. A latency of 20 cycles is used for the delay buffers. This is derived from current link hardware designs which features a 4-stage pipeline running at 150MHz at each end of the link. This pipeline assembles frames containing 8 packets which then takes 12 cycles to be sent over the S-ATA link for a total of 20 cycles.

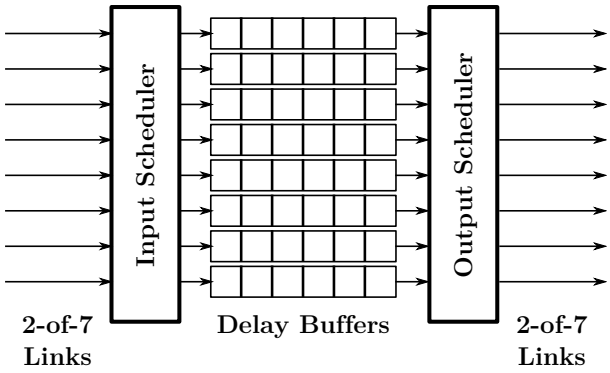


Figure 7: The model of a single direction of the S-ATA link used by the simulator. Every cycle, the input scheduler attempts to receive a packet from an incoming 2-of-7 link and place it in a corresponding delay buffer. Likewise, the output scheduler attempts to forward a packet which has reached the end of the delay buffer to its corresponding 2-of-7 link. The use of separate delay links prevent a single channel from blocking the others while the schedulers ensure relatively realistic behaviour. The delay approximates the latency incurred by assembling available packets into frames, transmitting them and disassembled again.

3.2 Simulator Design

Previous work made use of a simulation using the INSEE [7] framework. This simulator features a traffic generation component (TrGen) along with a simple functional network simulator (FSIN). FSIN unfortunately does not provide accurate timing information: all events are deemed to take one simulation cycle. This limitation prevents it's use in this experiment.

INSEE is alternative able to use the cycle-accurate 'SICOSYS' network simulator. Though this simulator provides high-accuracy timing information, it uses vast amounts more memory for simulation. The INSEE authors measured the use of around 350MB of memory for a 16-node system compared to around 20MB used by FSIN. This level of resource usage prevents the use of this simulator in this experiment where simulations involving thousands of nodes are required.

As a result of this unsuitability, a new event-based simulator was developed.

3.2.1 Parallel Simulation

The Parsec simulation environment [1] is a parallel event-based simulator which has been used for both network and circuit level simulations making it well suited to the SpiNNaker network which fits into the middle of this space.

In Parsec, 'logical processes' (LPs) emit timestamped events which are received by other LPs. These LPs each feature an 'earliest output time' (EOT) and 'earliest input time' (EIT) which indicate the lower-bound of timestamps of packets which could be transmitted received by an LP respectively.

An LP can run in parallel with another LP whose EIT is less than their EOT. This ensures that any events produced by the first LP have a timestamp which is more recent than the other LPs EIT and thus can be accepted. If this is not the case, causality is violated as the second LP may have processed an event with a timestamp equal to its EIT.

Parsec provides a framework for defining LPs and their EITs and EOTs along with a scheduling system which can potentially run the LPs in parallel where possible. The system can do this in one of three ways:

Conservative Scheduling will only allow LPs to run in parallel which cannot result in a causality violation and otherwise executes LPs are executed in serial.

Optimistic Scheduling tries to run LPs in parallel even if they may result in a causality violation. When such a violation is observed the system is wound-back to an earlier state to correct the violation.

Hybrid Scheduling uses a mixture of the two approaches allowing each to be used in parts of the system where it is most successful.

3.2.2 Serial Simulation

Unfortunately, due to time-constraints, Parsec was not used in the implementation of the simulator. A simple[3], serial event-based simulator based on the Verilog simulator [2] was built using Python.

In such a simulator the program is broken into blocks of code which are scheduled to occur at some point in time. When an event is 'emitted' by a segment of code, functions bound to this event are scheduled. Events may be emitted 'in the future' causing the bound functions to be executed at some point later in simulation time.

The scheduler consists of three queues as shown in figure 8. Functions are taken from the 'active queue' and executed. Execution may result in new functions being scheduled to occur immediately, at the very end of the cycle[4] or at some time in the future. Such functions are placed in the associated queue as shown in the figure.

Once the active queue is exhausted the contents of the inactive queue are moved to the active queue and execution resumes. This may result in new events being added to the inactive queue in which case the process repeats until both the active and inactive queues are empty.

Next all events from the next available timestep in the postponed queue are placed in the ready queue and the new cycle begins. The scheduler terminates once all three queues are

[3] A Verilog-style scheduler is extremely simple to implement requiring only 72 lines of Python (or 26 without white-space and inline documentation).

[4] In hardware description languages such as Verilog there is a notion of 'non-blocking' assignment. In such assignments the RHS is calculated immediately and the result later written to the LHS only at the very end of the cycle. For example, a register swap using non-blocking assignment could be written as `a<=b; b<=a;` where `<=` is the non-blocking assignment operator.

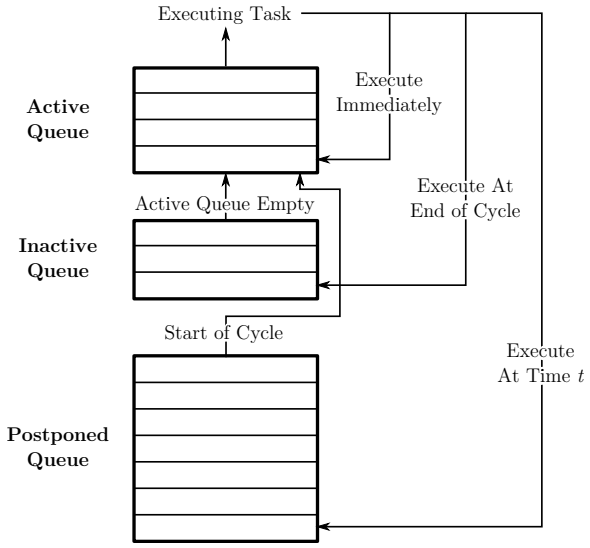


Figure 8: A Verilog-style scheduler.

depleted or after a user defined number of cycles has been executed.

4 Experimental Results

Experiments were conducted to validate the model's basic behaviour followed by experiments to observe the latency of packets in the system and emergency routing behaviour.

4.1 Confirmation of Topological Properties

The simulator was run for 20,000 cycles with emergency routing disabled on networks of various sizes and the average and maximum path length of packets recorded. The measured results are shown in table 1 alongside computed expected values.

The mean measured distances match those expected to 3 s.f. with the exception of the 48×48 system. This discrepancy may be due to the fact that in this larger system, packets taking the longest routes take around 935 simulation cycles. The simulation is terminated, not once all sent packets have arrived, but rather once a certain simulation time has been reached. As a result, the mean distance in the system is likely to be skewed in favour of packets taking less time to arrive.

Table 1: Path lengths in the simulated network: computed vs. measured.

Network Size	Mean		Maximum	
	Comp.	Meas.	Comp.	Meas.
12×12	5.653	5.646	9	9
24×24	10.326	10.290	17	17
48×48	19.663	19.119	33	33

The maximum distance observed matched the expected values for every network tested.

4.2 Communication Latency

The latency added by the S-ATA links can be seen in figure 9a where the minimum packet latency is plotted against a given number of hops in a 48×48 chip system. The system is shown using only 2-of-7 links, with realistic S-ATA links with latencies of around 68 cycles (as described in §3.1.3), and with exaggerated latencies of 248 cycles.

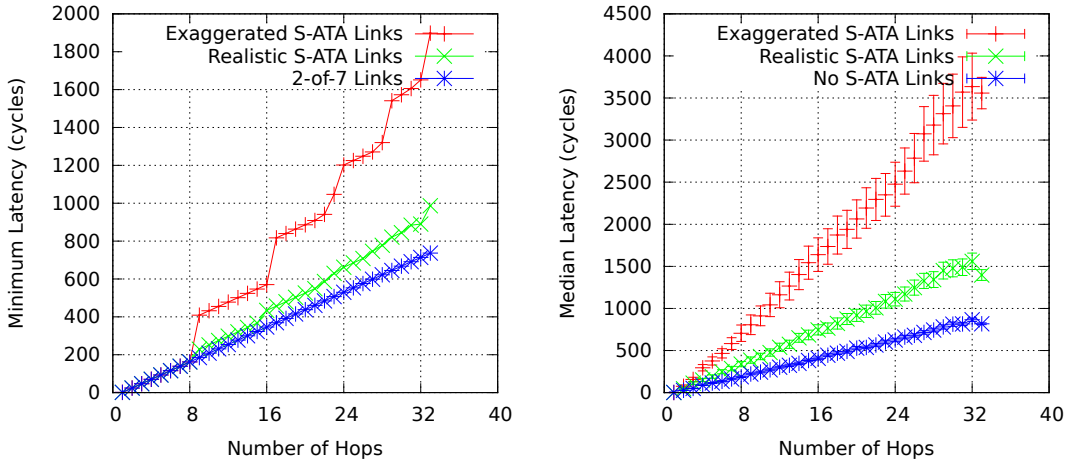
Steps can be seen every time the number of hops passes a multiple of 8, the number of chips in any single dimension on a board, after this a S-ATA link must be used resulting in the step in latency. For smaller numbers of hops, the steps are cleanly defined and have sizes which correspond to the difference between the S-ATA link modelled and a regular 2-of-7 link.

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

The median latency of an n-hop path increases smoothly as shown in figure 9b 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 S-ATA links result in an 80.4% latency overhead.

4.2.1 Latency Distribution

The packet latencies on systems without S-ATA links follow a power-law style distribution which can be seen in figures 10a and 10b. In systems using S-ATA links, the distribution becomes fragmented due to the slow links as shown in figure 10c. In larger systems, which



(a) Minimum latencies

(b) Median latencies, error bars = $0.1 \times$ IQR

Figure 9: Latency for packets travelling along paths of varying lengths when boards are connected using different types of link.

feature greater amounts of traffic, these fragments are hidden by noise added by routing delays and so the distribution returns to approximating a power-law as in figure 10d.

4.3 Dimension Order Routing Effects

The routing algorithm used relies on all 'hops' through the system taking the same amount of time in order to provide minimal-latency routes. This assumption can be visibly seen to break-down in systems using S-ATA links.

Figure 11a shows how the latencies vary across an idle system from a single point. It can be seen that where board boundaries are crossed there is a general increase in latency. Because of this, the contours are visibly distorted from their normal hexagonal shape. This is particularly visible at the bottom left and top right of the figure.

When the rest of the system is also loaded with other traffic, this effect appears to be further exaggerated as can be seen in 11b. This behaviour is likely to be caused by the increase in the usage of other channels in the S-ATA links increasing the latency for any given channel.

A step in latency is also visible within individual boards as can be seen in figure 12. Here there are clear edges where the dimension order routing traverses the dimensions in an order which incurs an extra board crossing.

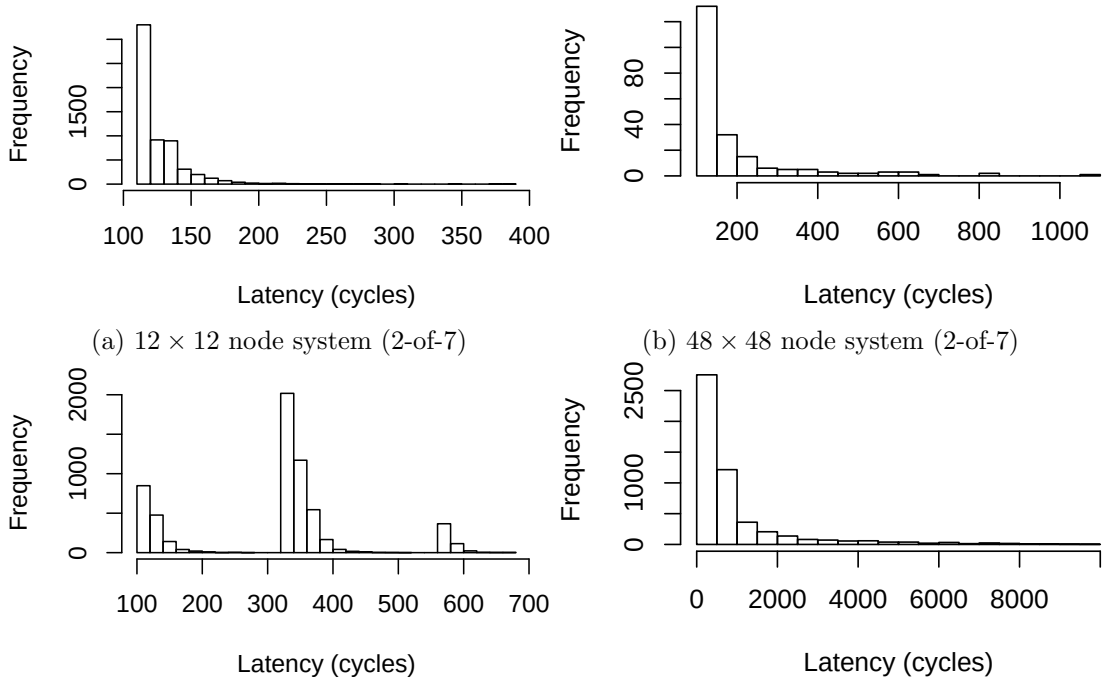
4.4 Emergency Routing

The 'emergency routing' feature of the SpiNNaker system was enabled and its usage is shown in figure 13. There are clearly visible hot-spots along horizontal edges of many of the boards.

If emergency routing is used while trying to cross north over a S-ATA link the packet will be diverted to the west node. Once at the west node, the router tries to send the packet north-east over the same busy A-ATA link and transmission is again delayed. This also increases congestion at that node which increases the likelihood emergency routing also being used. This pattern is a likely cause of the hotspots displayed in the figure.

5 Conclusions & Further Work

In this paper we have described the SpiNNaker architecture and its heterogeneous interconnection. A serial simulator was developed for a simplified model of SpiNNaker which includes a model of the in-development S-ATA inter-board links.



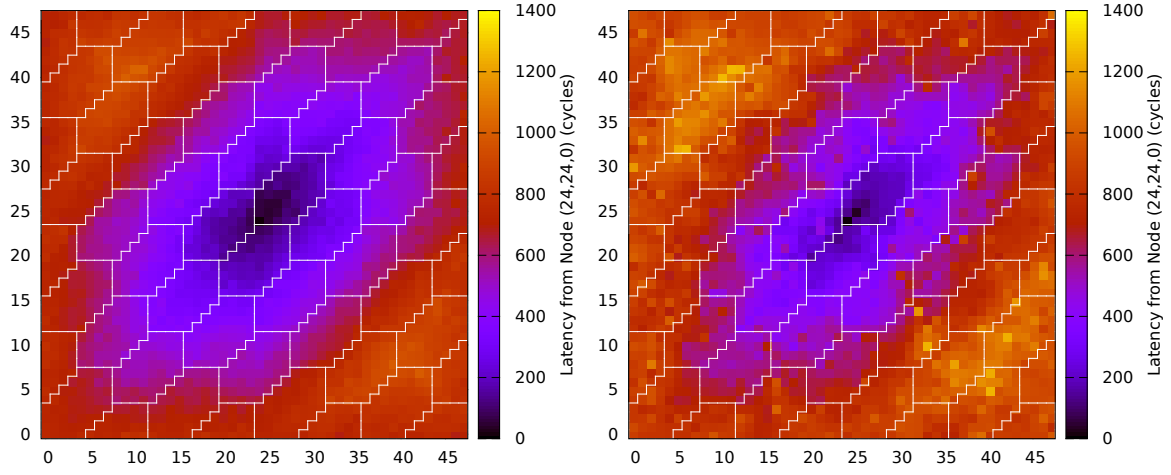
(a) 12×12 node system (2-of-7)

(b) 48×48 node system (2-of-7)

(c) 12×12 node system (S-ATA)

(d) 48×48 node system (S-ATA)

Figure 10: Histograms of packet latencies for 6-hop paths on different system sizes. (a) and (a) were simulated on a system using only 2-of-7 links. (c) and (d) were simulated on a system with boards connected via S-ATA links with exaggerated latencies.



(a) Unloaded System

(b) Loaded System

Figure 11: Heat-maps showing the packet latency from the central node (24,24,0) to the rest of the system. Each pixel represents a node in the system, the boundaries of boards are outlined in white. Note: A small number of randomly distributed nodes did not receive a packet from (24,24,0) during simulation due to packets not arriving within the simulation's execution. For clarity, the colour of these nodes has been interpolated.

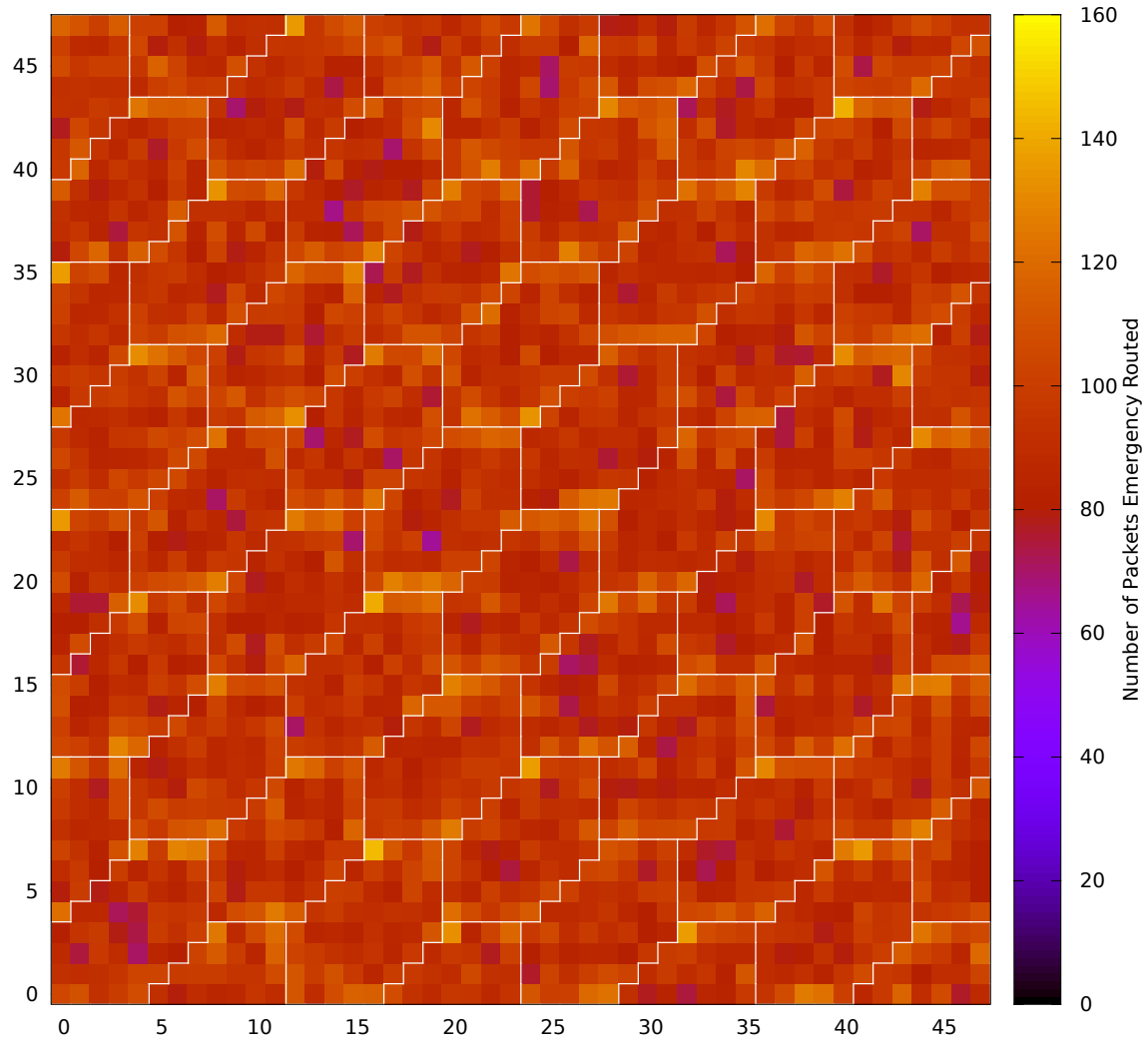


Figure 13: Heat-map showing the emergency-routing usage in a normally loaded system after 20,000 cycles. Hotter areas are visible along horizontal board edges.

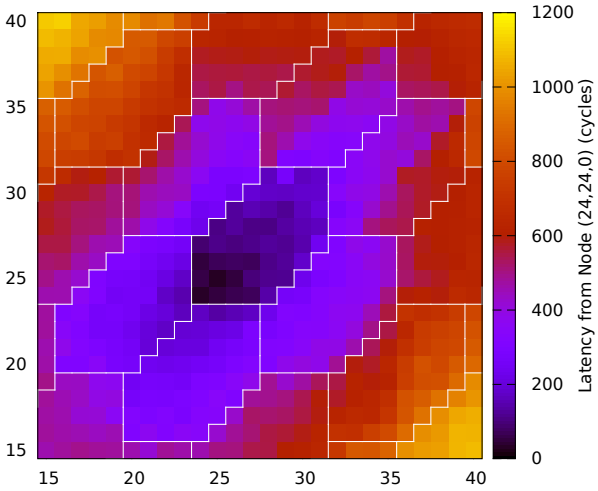


Figure 12: Heat-map of packet latencies from central node (24,24,0) on an unloaded system. Pixels are chips, white outlines are drawn along board outlines. The effects of border-unawareness in routing are visible as a step-change in latency along the diagonals of the upper-right boards. The S-ATA latency in this system has been exaggerated to aid visibility but is still present when realistic S-ATA latencies are used.

Our results have shown that the inclusion of S-ATA links has inevitably increased the latency of packet transmission. In particular, as the amount of traffic in the system increases so too does the effect of the inter-board boundaries.

Naive dimension-order routing makes no effort to minimise the use of expensive inter-board links. Experimental results show an 80% latency overhead to the median packet latency. Some of this latency can be attributed to the specific dimension order used for routing. Future work could investigate a small modification to the routing algorithm allowing alternative dimension orders to be used to reduce the number of borders crossed potentially reducing the median latency.

Finally, the 'emergency routing' facility provided by SpiNNaker unfortunately makes the assumption that links are independent. This assumption does not apply to the S-ATA links where multiple links share a single multiplexed path. As a result, emergency routing simply extends the path length a packet takes without improving performance. Future work may experiment with the effects of selectively disabling emergency routing for boundary links to avoid this effect.

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. Parsec: A parallel simulation environment for complex systems. *Computer*, 31(10):77–85, 1998.

[2] Doulos. *'Comprehensive Verilog' Course Book*. Doulos, 2008.

[3] 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.

[4] Jim Garside. Spinnlink frame transport. A draft/working document describing the S-ATA link.

[5] 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.

[6] L.A Plana, S.B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang. A gals infrastructure for a massively parallel multiprocessor. *IEEE Design & Test of Computers*, 24(5):454–463, September–October 2007.

[7] F. Ridruejo Perez and J. Miguel-Alonso. INSEE: an interconnection network simulation and evaluation environment. *Euro-Par 2005 Parallel Processing*, pages 643–643, 2005.

[8] Xilinx Inc. *Spartan-6 FPGA Family Product Brief*.