Chip-like Placement for Large-Scale Brain Simulations

Jonathan Heathcote

Short Report

1 Introduction

The human brain contains around 100 billion neurons connected by 100 trillion synapses [1]. Its scale makes the study of real brains extremely challenging and researchers are increasingly turning to computer simulations to gain greater visibility of its behaviour.

To handle the scale of these models both conventional supercomputers [2] and specialised computer architectures such as SpiNNaker [3] have been employed. These systems consist of a large number of computational nodes, typically CPUs, interconnected by a communications network as illustrated in figure 1. Due to the number of synaptic connections in biological neural networks, the communications network of a simulator can become a bottleneck when modelling large networks. As a result, careful use of network resources is critical to the performance of large-scale models. By placing connected neurons close together in the supercomputer's network, the network resource required to exchange messages is reduced.

Finding optimal solutions to the placement problem for arbitrary neural networks is an NP-complete problem. While some progress has been made towards constructing algorithms which find approximate solutions, they do not scale to the size of problems such as large-scale neural-modelling. A similar placement problem is faced by chip designers who must place circuits containing billions of transistors on to the surface of a chip. This project aims adapt the techniques used in chip circuit placement for placing neural simulations in supercomputers.

2 Current state of the art

Conventional large-scale supercomputer applications such as finite element modelling and molecular dynamics are partitioned and placed manually using domain knowledge of the problem [4]. In these applications the behaviour of materials is modelled by spatially dividing the material into pieces which are simulated independently as shown in figure 2. By exchanging information between neighbouring pieces of the simulation, the simulation proceeds coherently. Thanks to the regularity of this connectivity, optimal placement in many supercomputers is trivial. For example, for the torus network topologies used by the majority of the world's highest performing supercomputers [5], the topology of the network may exactly match the topology of the problem.

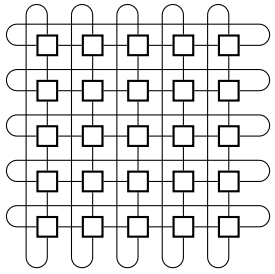

Figure 1: A 2D torus network. Squares represent computational nodes and lines are connections between them.

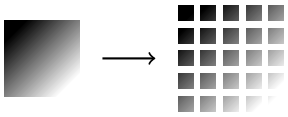

Figure 2: Partitioning of a 2D sheet into pieces for finite element modelling on a supercomputer.

Unfortunately, not all applications have such obvious placements. In industry, popular supercomputer management software such as SLURM [6] places applications arbitrarily within the supercomputer's network when a manual placement is not supplied. Some manufacturers – such as Cray – have added their own extensions to SLURM which use simple greedy heuristics to place jobs. Despite the apparent lack of interest in industry, researchers have demonstrated the significant effects placement quality can have on real-world applications [7]. This work has so-far focused on systems comprising tens or low hundreds of nodes [8, 9] or specific subsets of applications [10]. Existing placement algorithms are unable to scale to biological-

scale neural models and the complexity of the connectivity defies simple greedy approaches. As a result, new techniques must be sought.

3 Work to date

Chip designers have been tackling another instance of the placement problem for decades. Modern CPUs contain billions of components, each of which must be placed on the 2D surface of a chip while trying to keep connected components close together. Within this field many algorithms have been proposed and used successfully at scales meeting or exceeding those of many neural simulations [11]. Though not all algorithms used for chip placement are adaptable to non-2D-planar surfaces, a number are and adapting these has formed the focus of the work to date.

Current efforts target placement of neural simulations within SpiNNaker, a specialised computing platform for neural simulations [3]. Like other supercomputers, SpiNNaker consists of a large number of processing nodes (up to one million in the largest planned machine) connected via a toroidal interconnection network, similar to figure 1. The SpiNNaker architecture targets neural simulations containing up to a billion neurons and prototype hardware is readily available to the author making it an ideal candidate for experimentation.

Preliminary work included developing a software infrastructure [12], allowing experimental placement algorithms to be benchmarked against real SpiNNaker hardware and popular neural modelling software such as PyNN [13] and Nengo [14]. Subsequent work has focused on adapting chip placement software and benchmarking its performance against existing supercomputer placement approaches.

3.1 Placement

Three placement algorithms have been implemented which are representative of existing supercomputer and chip-like approaches:

rand A random placement algorithm which places neurons with no regard for their connectivity.

This trivial placer forms a baseline for placement performance and is analogous to the approach used by many supercomputer systems.

hilbert A greedy placement algorithm similar to that used in Cray super computers [6].

Pieces of an application are placed on sequential computational nodes along a Hilbert curve path through the machine, greedily inserting the next most heavily connected piece in each step.

This approach is representative of the more sophisticated placement approaches used in industry.

sa A simulated annealing [15] based placement algorithm based on the VPR chip placement software [16].

In simulated annealing, neurons are swapped randomly between nodes in the supercomputer. Swaps are reverted with some probability if they move connected nodes further apart. Over time, the probability of keeping bad swaps decreases until eventually only beneficial swaps are accepted. By initially accepting some detrimental swaps the algorithm is less likely to become trapped in local minima.

This approach is widely used for chip placement problems [11] and is representative of the approaches to placement in this field in terms of algorithm runtime and solution quality.

To accelerate development, all three algorithms were implemented in Python and consequently suffer a fixed penalty in runtime performance compared with commercial implementations. Critically, the computational complexity of these implementations is unaffected making relative comparisons amongst them meaningful. Further, despite the overhead, their runtime has proven acceptable for small to medium-scale placement problems involving around tens of thousands of computational nodes.

3.2 Routing

The selection of routes taken by messages in a supercomputer's network is known as the 'routing problem'. Alongside placement, routing is an important factor which partly determines the performance of a supercomputer's network. While outside the main scope of this research, the choice of routing algorithm for use in experimental work deserves due consideration.

Though many general-purpose routing algorithms are available [17], only a fraction specifically target *multicast* routing where messages may be sent to multiple destinations, as is the case for neural simulations. For this work, the Neighbourhood-Exploring Routing (NER) algorithm was selected which specifically targets neural applications running on SpiNNaker [18].

The basic NER algorithm has been extended to work around faults in the network which are inevitable in any large scale real-world system.

Additionally, a novel formula for finding paths through the network topology used by SpiNNaker was developed. This formula was used in the kernel of the implementation of the NER algorithm and, unlike published formulae to date, evenly loads available network resources [19].

3.3 Benchmarking

Though suites of standardised benchmarks are widely available for both chip design [20] and generic supercomputer applications [7], no such benchmarks exist for large-scale neural applications. A representative set of neural application benchmarks is being assembled as part of this work to address this shortfall.

The benchmark suite assembled so far contains a mix of synthetic benchmarks and real-world neural models of various sizes. The synthetic benchmarks are all arbitrarily scalable allowing comparisons of placement quality at a wide range of problem sizes. The real-world models are taken directly from the literature and give a concrete indication of placement quality in practice.

3.4 Results

To assess the performance of the three placement algorithms, artificial network traffic closely mimicking the traffic produced by the benchmark neural applications is fed into SpiNNaker's interconnection network. By measuring the maximum intensity of traffic the network can handle, the placement algorithms' relative performance can be compared. A placement which can sustain greater network traffic intensity corresponds with higher speed, or more detailed neural simulations.

On average, the **sa** placement algorithm produced placements which could support three times more throughput than the baseline **rand** placement and double the throughput of the supercomputer-style **hilbert** placer. A subset of the benchmarks which mimic the communication found in thalamocortical neural models (for example, the model used by Davies *et al.* [21]), sustain around 30 times more throughput when placed by the **sa** algorithm than either **rand** or **hilbert**.

The major shortcoming encountered with the **sa** algorithm so far is that, while **rand** and **hilbert** can place benchmarks consisting of tens of thousands of nodes in seconds and minutes respectively, the runtime of **sa** reaches around five hours on a typical workstation for benchmarks of the same size.

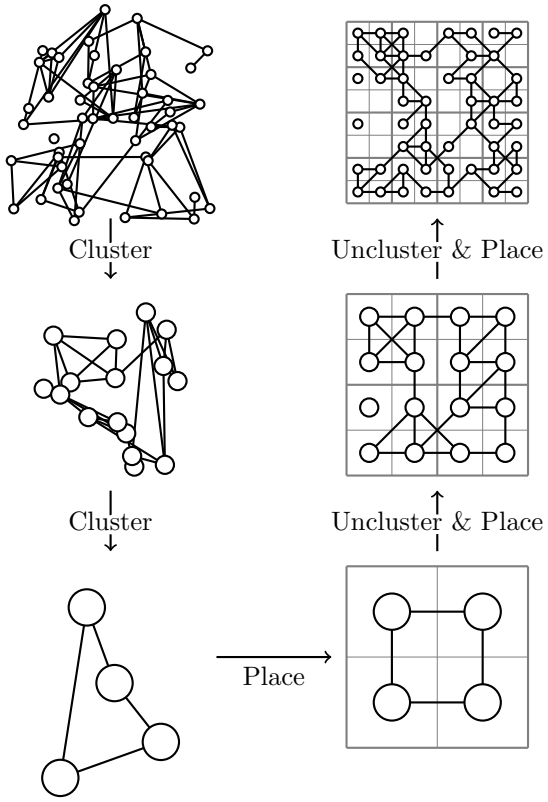

Figure 4: Hierarchical clustering based placement.

4 Research plan

The remaining work in this project focuses primarily on reducing the runtime of the **sa** placement algorithm and extending it to other supercomputer architectures. Figure 3 gives projected schedule for the remaining work.

4.1 Hierarchical clustering

Hierarchical graph clustering techniques are widely used to extend the scalability of circuit placement algorithms [22]. As illustrated in figure 4, by recursively clustering the problem graph a smaller placement problem is obtained which can be placed inexpensively. Each cluster is then individually unclustered and placed to arrive at a solution to the original placement problem. For placers with large computational complexity with respect to problem size, clustering dramatically reduces runtime by decomposing problems into a large number of much smaller problems.

By using a clustering-based approach for the **sa** algorithm, a significant reduction in runtime can be expected, while maintaining similar placement quality. Preliminary work on graph clustering suggests that placement problems involving millions of computational nodes can be reduced to hundreds of

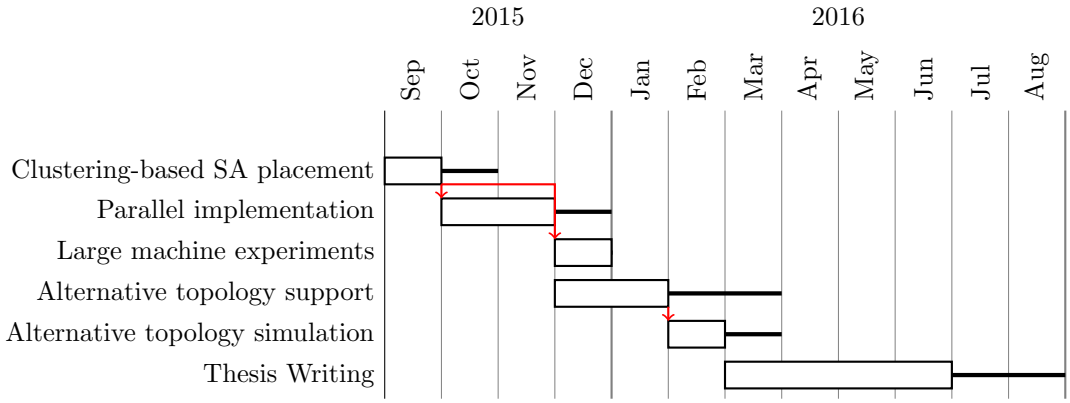

Figure 3: Research plan Gantt chart. Boxes indicate the expected duration of a task, thick lines indicate slack and red arrows show dependencies between tasks.

nodes in minutes on a workstation computer. Such a problem could then be solved by **sa** in a matter of seconds.

4.2 Parallel implementation

Previous attempts to achieve speed up from parallelisation in simulated annealing algorithms have reported extremely limited or even negative speed-up [23]. A hierarchical clustering based simulated annealing approach brings new opportunities for parallel implementations. In the unclustering phase of the algorithm a large number of small, independent placement problems are generated which can be trivially executed in parallel. An implementation of this parallel approach will be developed suitable for execution on SpiNNaker itself.

Notably, unlike in the field of circuit-placement, by using the target supercomputer to execute the placement algorithm, the amount of parallel resource is guaranteed to scale with the size of the placement problem. This unique property means that for a clustering based approach, the computational resource available will always be greater than the amount required by the algorithm by a factor determined by the degree of clustering employed. Any remaining computational resource can then be absorbed using more traditional low-gain techniques for paralleling simulated annealing. For example, multiple competing instances of the algorithm may be used in parallel with different random seeds and the best result selected [24].

4.3 Large-scale experiments

In the coming months, a $\frac{1}{2}$-scale SpiNNaker machine with 518,400 cores will be brought on-line allowing the extension of the benchmarking work to large scale systems. In principle, this process should be entirely automatic as all necessary infrastructure has already been developed.

4.4 Alternative network topologies

The placement algorithms will be extended to support other torus- and tree-based network topologies found in supercomputers. Experiments using the same benchmarks will be repeated on these alternative topologies using the INSEE [25] network simulator which is capable of simulating large scale supercomputer networks [26].

5 Conclusions and future research

The scale of the placement problem for neural applications renders existing supercomputer placement algorithms insufficient. In this project, placement algorithms used by the chip design community have been adapted and extended to support the placement of supercomputer jobs with a focus on neural applications. Upcoming work will further extend these algorithms to support massively parallel execution exploiting the inherent availability of parallel computational resources when working on supercomputer applications. Further, the techniques developed will be generalised to support other supercomputer architectures and the performance across these architectures assessed.

Though current work has focused on placing neural models, these techniques may also be applicable to other supercomputer applications such as electronic circuit simulation. Additionally, as system-on-chip architectures grow in scale and begin to look increasingly like supercomputer networks, the need for intelligent placement at large scales will become more important there too.

References

[1] Robert W Williams and Karl Herrup. The control of neuron number. *Annual review of neuroscience*, 11(1):423–453, 1988.

[2] Rajagopal Ananthanarayanan, Steven K Esser, Horst D Simon, and Dharmendra S Modha. The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12. IEEE, 2009.

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

[4] Jonathan R Clausen, Daniel A Reasor, and Cyrus K Aidun. Parallel performance of a lattice-Boltzmann/finite element cellular blood flow solver on the IBM Blue Gene/P architecture. *Computer Physics Communications*, 181(6):1013–1020, 2010.

[5] Hans Meuer. Top500 List - June 2015. `http://www.top500.org/list/2015/06/`, June 2015. Accessed: 23 July 2015.

[6] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[7] Javier Navaridas, Jose Antonio Pascual, and Jose Miguel-Alonso. Effects of job and task placement on parallel scientific applications performance. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 55–61. IEEE, 2009.

[8] Hu Chen, Wenguang Chen, Jian Huang, Bob Robert, and Harold Kuhn. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 353–360. ACM, 2006.

[9] Emmanuel Jeannot, Guillaume Mercier, and François Tessier. Process placement in multicore clusters: Algorithmic issues and practical techniques. *Parallel and Distributed Systems, IEEE Transactions on*, 25(4):993–1002, 2014.

[10] Emmanuel Jeannot and Guillaume Mercier. Near-optimal placement of MPI processes on hierarchical NUMA architectures. In *Euro-Par 2010-Parallel Processing*, pages 199–210. Springer, 2010.

[11] Gi-Joon Nam and Jingsheng Jason Cong. *Modern circuit placement: best practices and results*. Springer, 2007.

[12] Jonathan Heathcote, Andrew Mundy, and Jamie Knight. Rig. `https://github.com/project-rig/rig`, 2015.

[13] Andrew P Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2, 2008.

[14] Andrew Mundy, Jamie Knight, Terrence C. Stewart, and Steve Furber. An efficient SpiNNaker implementation of the Neural Engineering Framework. In *IJCNN 2015*, 2015. Proceedings not yet publicly available.

[15] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[16] Vaughn Betz and Jonathan Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic and Applications*, pages 213–222. Springer, 1997.

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

[18] Javier Navaridas, Mikel Lujn, Luis A. Plana, Steve Temple, and Steve B. Furber. Spinnaker: Enhanced multicast routing. *Parallel Computing*, 45(0):49 – 66, 2015.

[19] Jonathan Heathcote. Finding shortest paths in hexagonal torus topologies. `http://jhnet.co.uk/articles/torus_paths`, 2015. Accessed 23 July 2015.

[20] Gi-Joon Nam, Charles J Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proceedings of the 2005 international symposium on Physical design*, pages 216–220. ACM, 2005.

[21] Sergio Davies, Javier Navaridas, Francesco Galluppi, and Steve Furber. Population-based routing in the spinnaker neuromorphic architecture. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[22] Charles Alpert, Andrew Kahng, Gi-Joon Nam, Sherief Reda, and Paul Villarrubia. A semi-persistent clustering technique for VLSI circuit placement. In *Proceedings of the 2005 international symposium on Physical design*, pages 200–207. ACM, 2005.

[23] Adrian Ludwin, Vaughn Betz, and Ketan Padalia. High-quality, deterministic parallel placement for FPGAs on commodity hardware. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, pages 14–23. ACM, 2008.

[24] Malay Haldar, Anshuman Nayak, Alok Choudhary, and Prith Banerjee. Parallel algorithms for FPGA placement. In *Proceedings of the 10th Great Lakes symposium on VLSI*, pages 86–94. ACM, 2000.

[25] 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.

[26] 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.

A Thesis outline

The following section-level outline gives the planned thesis structure for this project. Sections which are reliant on upcoming work are indicated with a dagger (†);

1. Introduction
2. Background
    (a) Supercomputer architectures
    (b) Supercomputer applications
    (c) Neural modelling and simulation
    (d) Neuromorphic simulators
    (e) The SpiNNaker architecture
    (f) The impact of placement
3. Related work
    (a) Supercomputer placement
        i. Manual placement
        ii. Greedy placement
        iii. Placement as a Quadratic Assignment Problem
        iv. System partitioning
    (b) Chip placement
        i. TreeMatch
        ii. Quadratic placement
        iii. Non-linear placement
        iv. Partitioning-based placement
        v. Simulated annealing
        vi. Clustering
4. Placement
    (a) Adaptations for neural simulation
    (b) Adaptations for supercomputer architectures
    (c) Hierarchical clustering †
    (d) Parallelisation †
5. Benchmarks
    (a) Statistical neural connectivity models
    (b) Thalamocortical column models
    (c) Convolution networks †
    (d) Pipeline-style architectures
    (e) Learning (plastic) models †
    (f) Artificial (classical) neural networks †
    (g) Experimental setup
6. Results
    (a) Algorithm runtime
    (b) Parallel execution speed-up †
    (c) Static analysis of placement quality
    (d) Empirical results
        i. Small-medium SpiNNaker experiments
        ii. Large-scale SpiNNaker experiments †
        iii. Torus topology simulations †
        iv. Tree topology simulations †
7. Discussion
    (a) Cost/benefit of chip-style placement †
    (b) Approach scalability †
    (c) Generalisability †
8. Future research
    (a) Structural plasticity in neural networks
    (b) Alternative placement problems
    (c) Applicability to system-on-chip
9. Conclusions
10. Appendix: Routing
    (a) Supercomputer network routing
    (b) Applicability of chip-design approach
    (c) Neighbourhood exploring routing
    (d) Fault-avoidance
    (e) Finding short paths in hexagonal torus topologies