SpiNNaker: The design automation problem

Andrew Brown[1], David Lester[2], Luis Plana[2], Steve Furber[2] and Peter Wilson[1]

[1] Electronics and Computer Science, University of Southampton, UK
[2] Computer Science, University of Manchester, UK
{adb,prw}@ecs.soton.ac.uk
{drl,lap,sbf}@cs.man.ac.uk

Abstract. This paper describes the design automation issues and techniques used to design a massively parallel processing platform – SpiNNaker – from a hardware and systems design perspective. The emphasis of this paper is addressing the key problem of resource mapping, where multiple threaded programs are to be targeted onto a hardware platform that consists of multiple ARM cores and other resources such as memory and networks. In addition, the design environment is considered to ensure that a designer can program applications onto this environment in a practical manner.

1 Introduction

SpiNNaker is a massively parallel multi-core computing engine, consisting of a vast array of ARM cores and a fast interconnect fabric. Although strictly a clocked system, each ARM core is effectively decoupled from its peers, and individual processors communicate with each other by means of packets; a packet incident upon a processor causes an interrupt which handles the packet. The system has been designed to support the real-time simulation of large aggregates of spiking neurones. The development strategy is coarsely incremental, but the final goal is to be able to simulate aggregates of a billion neurones, where each of a million processors is supporting the emulation of a thousand neurones.

Various aspects of the system (physical architecture and interconnect fabric, neural and synaptic models) have been described in detail elsewhere [1-3]. The overall structure of the system is shown in figure 1. This paper describes some of the problems associated with mapping the abstract neural connection topology (which may, but is not required to be, three dimensional) onto a physical two dimensional array of processors. The requirements are highly analogous to the automated place and route (APR) problem experienced in chip design, and the evolution of those problems almost identical. In the world of IC design, components representing the realisation of a circuit (transistors, resistors, vias and so on) have to be laid out on a two dimensional silicon die, and then the geometry of the interconnect defined. In the early days of IC design, this could be done by hand, but as the size and complexity of systems grew, design automation - initially a luxury - became a necessity. Today, it is simply not possible to lay out a state-of-the-art IC (5mm x 5mm die, feature size O(100nm)) by hand. So it is with SpiNNaker: the neural systems we wish to simulate are vast topologies of interconnected neurones, which have to be mapped onto our array of processors. The situation is actually worse than in the electronic counterpart:

in electronic designs, design automation tools and techniques capitalise heavily on a hierarchical input description, an advantage that is largely denied us in the current problem domain. This paper describes the development strategy for a suite of APR tools designed to be used to load the SpiNNaker data structures with large ($O(10^9)$) interconnected neurones.

II. THE HARDWARE PLATFORM

A. The chip

The internal architecture of the SpiNNaker chip is depicted in figure 1. Each chip contains 20 ARM processors (with a small amount of local memory in a Harvard configuration) and 6 bi-directional inter-chip link ports. These are interconnected by an on-chip network. External interfaces are also provided to a single bank of chip-local DRAM and Ethernet.

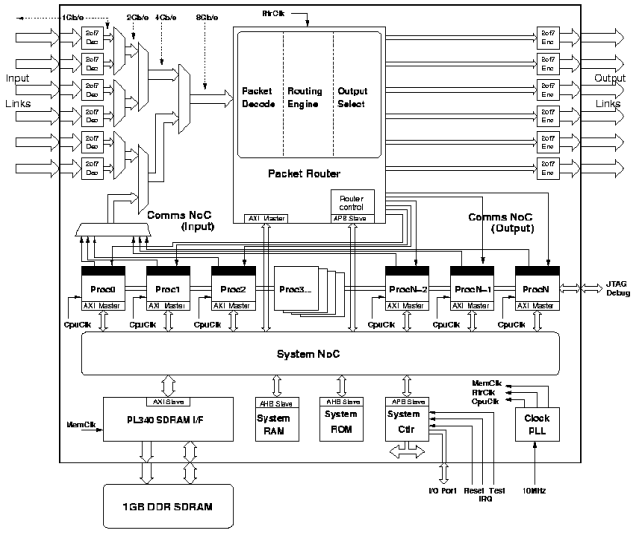

Figure 1: A single SpiNNaker node internals

The chips themselves are connected (via the inter-chip link ports) in a hexagonal mesh, mapped onto the surface of a toroid. This conveniently avoids edge effects, although it requires that the notion of geometric hop distance be handled rather carefully. However, this particular design decision is one of the more easily overturned: The economics of getting silicon right first time are significantly different to changing the inter-chip layout on a PCB.

Neurones are mapped statically onto an individual ARM processor, and the internal state of the neurones mapped onto each die is held in the associated DRAM. When a spike arrives at a processor, it fetches the state of the relevant neurone from the DRAM, processes the incoming spike, updates the state of the neurone in the DRAM and may broadcast spikes of its own.

B. Neural events (spikes)

Spikes are embodied by a 32-bit packet, a technique known as Address Event Representation (AER). When a neurone spikes, the spike is a pure asynchronous 'event': all of the information is conveyed solely in the time of the spike and the

identity of the neurone that emitted the spike. In a real-time system time models itself (possibly with a scaling factor, but we normally assume no scaling), so in an AER system the identity ('address') of a neurone that spikes is simply broadcast at the time that it spikes to all neurones to which the spiking neurone connects.

In SpiNNaker AER is implemented using packet-switched communication and multicast routing. Although the communication system introduces some temporal latency, provided this is small compared with biological time constants (which in practice means provided it is well under 1ms) then the error introduced by this latency is negligible.

III. PLACE AND ROUTE

A. A traditional methodology

Real circuits, though they may have hugely different functionalities, are all remarkably similar from a purely topological perspective. Most logic elements have a handful of inputs; most nets are connected to very few components, but a very few nets will be massively connected. As the general place and route problem is NP-complete, APR tools today all use heuristics to deliver the results. Whilst these are very powerful, and have been honed over several decades of use and development, a heuristic wins because it capitalises on the statistical properties of its input dataset. If you change the nature of the presented problem, the efficacy of a given algorithm must be viewed with some caution. However, there exists a vast and mature body of work in this area, and it would be foolish not to utilise it. The APR problem is conventionally broken down into the sub-problems of *placement*, *global* and *detailed* routing.

1. Placement

Placement (in the context of SpiNNaker) involves choosing a mapping between the neurones of the abstract topological circuit and the fixed geometry of the processor array. This placement is only weakly influenced by the properties of the interconnect. Two broad classes of placement algorithms exist: iterative and constructive. Iterative systems operate by starting with a very crude, computationally cheap (often both random and unrealizable), and improving upon it in an incremental fashion. Many different convergence criteria have been studied.

Force-directed placement (iterative)[4] considers the neurones (modules) as point masses, and the interconnect as springs, of weighted force constant. The system is allowed to relax to a configuration of minimum energy. This algorithm can be made extremely fast; although it is derived from the laws of physics, there is, of course, no necessity to abide by them if it is computationally inconvenient.

Simulated annealing (iterative)[5] also attempts to minimise the overall 'energy' of a system, by a sequence of random perturbations, the probability of acceptance of each depending on the improvement wrought as a consequence.

2. Routing

The routing problem is concerned with finding a route *between* a set of points, *round* a set of obstructions (placed modules) on a two-dimensional plane. The

routing problem is generally decomposed into *global* and *detailed* routing, the difference being the granularity of the analysis.

Routing algorithms are even more diverse and numerous than placement, and offer the usual spectrum of reliability and quality-of-solution vs. speed. One of the earliest - and possibly most versatile - is a graph-searching algorithm, known as Lees' maze-runner[6]. The algorithm can be used at arbitrary levels of granularity, and is guaranteed to find a solution for a given *single* route if it exists. On the other hand, it is relatively slow, and the overall success in finding a solution for a *set* of routes is not guaranteed. Numerous enhancements to reduce memory footprint and runtime exist, but the algorithm in its simplest form is widely applicable to all manner of graph-searching problems.

B. SpiNNaker-specific issues

The APR problem in the context of the SpiNNaker system may be summarized by figure 2: a fragment of circuitry, comprising six neurones, is to be mapped onto the fixed array of processing nodes. Each node can accommodate around 1000 neurones. Having decided upon this mapping, the routes (the sequences of nodes between source and target) must be established, and the routing tables defined in each node. What is different about the SpiNNaker context?

1. Data size

Conventional processing speeds have increased by many orders of magnitude over the past few decades, but even this is not enough to overcome polynomial complexity in an algorithm when the input datasets become large. APR of electronic circuits containing a billion components is feasible today in reasonable timescales, *but* these circuit descriptions are highly hierarchical, the decomposition being determined and fixed by human input.

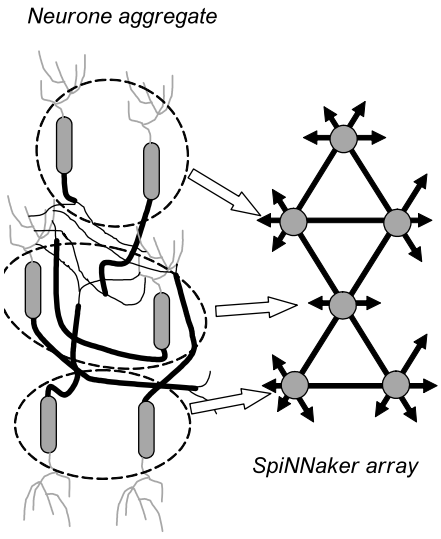

Figure 2: Fascicle mapping

The discussion of where we actually get (meaningful) circuits of a billion neurones has yet to be published, but irrespective of whether these circuits are generated semi-automatically or stochastically generated, the APR task will be formidable.

- 32 bit machines can only address four billion memory locations; it will not be possible to even hold (let alone process) the entire datastructure at one time in an APR machine.
- Even $O(n^3)$ algorithms - generally considered to be acceptable for APR problems - will be unusable.

The unavoidable outcome of these points is that aggressive

hierarchical decomposition of the neural aggregate descriptions is an absolute necessity.

2. Chip level topology

The SpiNNaker chips have been designed with six bidirectional I/O ports each, lending themselves naturally to a hexagonal placement on a two-dimensional plane. Six links were chosen to support a measure of fault tolerance in the final system, by providing a simple triangular bypass to each link if one fails. Identifying opposing edges of the array of chips folds the system naturally onto a toroid.

Point-to-point packet routing only requires routing table entries in certain nodes. Consider figure 3: a packet is to be sent from node S to node T; the route has been chosen to be S-1-2-3-4-T. The only nodes that require a routing table entry are S (this is *from* me), 2 (turn a corner) and T (this is *for* me). Packets incident on nodes with no corresponding routing table entry are simply passed through in a 'straight line' - the default route.

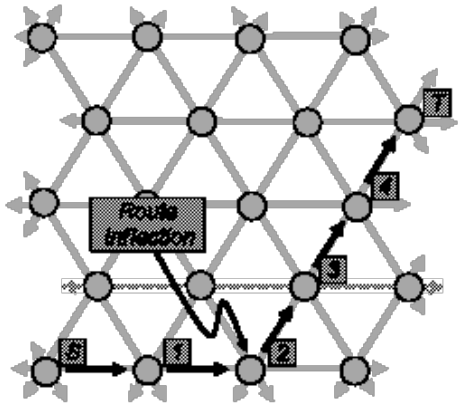

Figure 3: Chip level SpiNNaker interconnect

It is necessary to define some terminology before going further: A **fascicle** is the collective noun for a set of neurones. Other than to note that no neurone can be a member of more than one fascicle, the term *defines* nothing about any connectivity. There is an implication (and an efficiency assumption) that the neurones in a fascicle share common input fascicles (probably sparsely connected) and common output target fascicles (again, probably sparsely connected). There is no implication that they are interconnected with each other, though they can be (or not). However, none of this is *required* - it just makes the datastructure packing more efficient if it holds.

C. The framework

A **Fascicle Processor** is a single physical ARM core on a SpiNNaker chip. It may be host to zero, one or more fascicles in the simulation process. The intention is that of the *n* ARM cores on a chip, *n-1* will be Fascicle Processors. (The other ARM will be used for housekeeping functions.)

The overall APR structure is fairly conservative, and outlined in figure 4. It is a heuristic; the size of the datasets makes iteration a very expensive operation, and so the design intention is that a neural circuit will pass through the design flow only once. Feedback of any kind is to be avoided, but if this is not possible, the feedback loops have been arranged in order of computational expense:

fb1: Should never be necessary anyway.
fb2: Is cheap and may never even be necessary - see later.
fb3: Is cheap, but the usefulness is dubious.

fb4: The Loop of Last Resort: expensive, but allows the system to expand onto unused SpiNNaker chip sites if any exist.

1. Neurone to fascicle mapping

Here we take the input neurone circuit (figure 2, for example), and partition it between the Fascicle Processors. In outline, the algorithm - based on the Kernighan-Lin partitioning scheme[7] - is as follows:

1. The input graph G is bisected randomly (in terms of synapse count) into two subgraphs, A and B. (These are potential fascicles - we note that, in general, each will be far too big to fit into a single SDRAM, certainly for the first few recursion levels of the algorithm).
2. Define some size limit, h, corresponding to the approximate capacity of a Fascicle Processor.
3. Find the neurone (in A or B) that (a) is unflagged, and (b) that would make the biggest improvement (which may, actually, be the smallest degradation) to the penalty function d() if it were to be moved to the opposite subgraph.
   If Δd() is an improvement, and does not violate the fascicle size limit h, **then** {move it and flag it, return to the start of step 3}
   // Only here if the best Δd() is actually a degradation
   If it was the first attempt (i.e. no neurones are flagged) **then** stop, **else** clear flags and return to the start of step 3.
4. Recursively apply step 3, replacing G with A and B at each level. The size of G will (approximately) halve at each recursion level; a recursive branch can terminate when h < SDRAM size (i.e. we have created a fascicle that will fit into a node SDRAM).

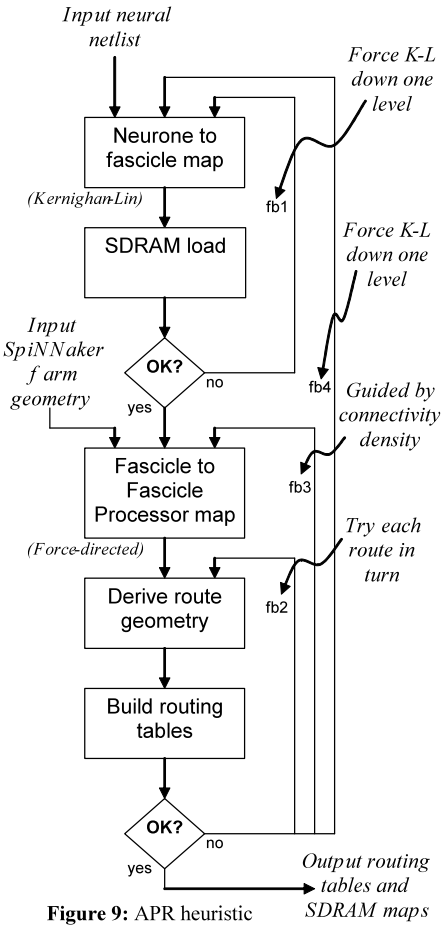

Figure 9: APR heuristic

The nature of the penalty function, d(), is worthy of some comment. In the traditional (electronic) context, it will represent the number of interconnects that cross the subgraph partition (i.e. pass between A and B). Here, however, because of the way the data is packed into the SDRAM, we are not attempting

to *minimise* the cut interconnects, we are attempting to *even out* the density of fascicle-fascicle interconnect. In essence, we look at each neurone in turn (subject to the restrictions in step 3 above), and see what effect would be had on the cut-line count if it were to be moved to the opposing subgraph. The neurone that is chosen is the one that minimises the total standard deviation of the cut-line counts.

2. SDRAM data

The output from the algorithm of the previous section is a set of bitmaps, which represent the connectivity of the neural circuitry, plus the state. These are loaded into the SDRAM of figure 1.

3. Fascicle to fascicle processor mapping

Having partitioned the neural network into fascicles, it is now necessary to map these onto the individual fascicle processors, as in figure 2. The partitioning achieved by the last algorithm simply divided the neural aggregate up into fascicles, but the attributes of geometric position were not assigned to the neurones. This is done using a combination of Lees algorithm and force-directed placement.

The force-directed algorithm is ideal for this; it has approximately linear complexity and delivers an acceptable placement extremely quickly

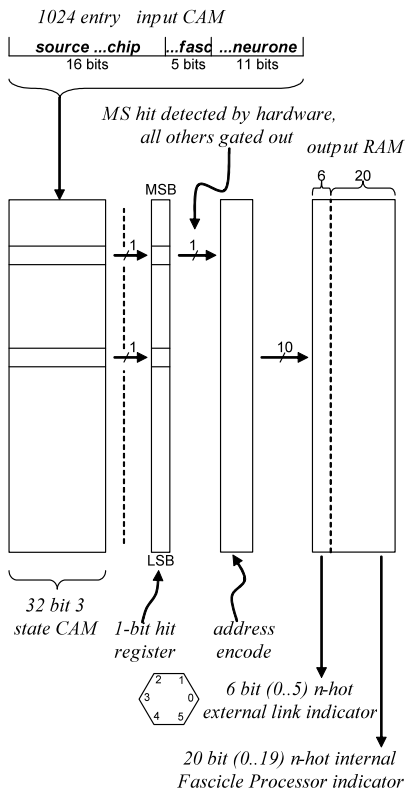

Figure 5: Routing tables

4. Routing geometry

The derivation of the individual packet trajectories over the system is simple. For a given point-to-point route, there will be a maximum of six 'shortest' routes (recall that the toroidal layout has no edges) and an arbitrary number of longer possibilities. The only real selection criterion here is the capacity of the routing tables at the inflection nodes. Each route is examined in turn, and that with the *lowest line integral of routing table occupancy* chosen[8]. Thus the interconnect route density is kept as even as possible over the routing surface.

5. Routing tables

The final step in the process of loading the SpiNNaker ensemble is the generation of the data to be contained in the routing tables. The structure of the routing table hardware is outlined in figure 5. The 32-bit source key is input to a 1024 x 32 bit tristate CAM. As it is a tristate (0, 1, X) CAM, in general, multiple hits will be both possible and common. These hits are written to a 1024 x 1 bit hit register. All but the most significant single bit in this register are discarded, and this single remaining bit treated as a 1024 bit 1-hot and passed into an address encoder. This generates a 10 bit binary equivalent, which drives a 1024 x 26 bit lookup RAM. The 26 bit word so

generated consists of a 6-bit nibble and a 20-bit nibble. The 6-bit nibble represents an n-hot external link indicator (0-5) to which the packet is forwarded (for example 010110 would cause the packet to be routed to external links 1, 2 and 4). The 20-bit nibble represents an n-hot internal Fascicle Processor address (0-19) to which the packet will be forwarded, triggering an interrupt as it arrives. (For example, 00001000100100000000 will cause packets to go to Fascicle Processors 8, 11 and 15 *on the current chip*. It is easy to see how packets may be duplicated by this mechanism. The 1024 x 32 bit tristate CAM is implemented as a 1024 x 32 bit binary CAM and a 1024 x 32 bit binary RAM. The RAM simply holds a bit mask indicating the position of the "don't cares" in the CAM. The corresponding bits in the CAM will *actually* be '0' or '1', but will never be read.

IV FINAL REMARKS

The design, development and realisation of a customised APR tool suite for the SpiNNaker system is a significant undertaking, requiring resources comparable to that of the hardware design. Like the hardware, the system is not yet complete, although the hope and intention is that the alpha release will coincide with the delivery of the prototype silicon.

The detailed design and development of the SpiNNaker hardware is an extremely complicated piece of electronic design, and this naturally makes heavy use of EDA tool suites. A simulation model of the chip has been built, but unfortunately (perhaps not unsurprisingly) this is unable to cope with the simulation of neural systems of any realistic size or complexity. It should be noted, however, that the datastructures required to support a bespoke behavioural simulator are all present in the tools described here; the addition of a behavioural simulation capability is not seen as a vast undertaking.

ACKNOWLEDGEMENTS

The SpiNNaker project is supported by the Engineering and Physical Sciences Research Council, UK, ARM and Silistix.

REFERENCES

[1] S.B. Furber, S. Temple, A.D. Brown, "On-chip and inter-chip networks for modelling large-scale neural systems" in *Proc. International Symposium on Circuits and Systems, ISCAS-2006*, Kos, Greece, May 2006.

[2] S.B. Furber and S. Temple, "Neural systems engineering", *J. R. Soc Interface*, **4**, no 13, pp 193-206 Apr 2007.

[3] L.A. Plana, S.B. Furber, S. Temple, M.M. Khan, Y. Shi, J. Wu and S. Yang, "A GALS infrastructure for a massively parallel multiprocessor", *IEEE Design and Test of Computers*, **24**, no 5, pp 454-463, Sept-Oct 2007.

[4] N.R. Quinn "The placement problem as viewed from the physics of classical mechanics", *IEEE Circuits and Systems*, **CAS-26**, no 6, pp 173-178, 1979.

[5] M.P. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing", *IEEE Transactions on Computer-Aided Design*, **TCAD-2**, no 4, pp 215-222, 1983.

[6] C.V. Lee, "An algorithm for path connections and its applications", *IRE Transactions on Electronic Computers*, pp 346-364, 1969.

[7] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", *Bell Sys. Tech. Journal*, **49**, pp 291-308, 1970

[8] A.D. Brown, "Automated placement and routing", *Computer-aided design*, **20**,