Event-Driven Configuration of a Neural Network CMP System over a Homogeneous Interconnect Fabric

M.M. Khan*, J. Navaridas†, A.D. Rast*, X. Jin*, L.A. Plana*, M. Luján*,
J.V. Woods*, J. Miguel-Alonso† and S.B. Furber*
***School of Computer Science, The University of Manchester, UK**
†University of The Basque Country, Spain
email: khanm@cs.man.ac.uk

***Abstract*—Configuring a million-core parallel system at boot time is a difficult process when the system has neither specialised hardware support for the configuration process nor a preconfigured default state that puts it in operating condition. SpiNNaker is a parallel Chip Multiprocessor (CMP) system for neural network (NN) simulation. Where most large CMP systems feature a sideband network to complete the boot process, SpiNNaker has a single homogeneous network interconnect for both application inter-processor communications and system control functions such as boot load and run-time user-system interaction. This network improves fault tolerance and makes it easier to support dynamic run-time reconfiguration, however, it requires a boot process that is transaction-level compatible with the application's communications model. Since SpiNNaker uses event-driven asynchronous communications throughout, the loader operates with purely local control: there is no global synchronisation, state information, or transition sequence. A novel two-stage "unfolding" boot-up process efficiently configures the SpiNNaker hardware and loads the application using a high-speed flood-fill technique with support for run-time reconfiguration. SystemC simulation of a multi-CMP SpiNNaker system indicates an error-free CMP configuration time of 1.3 ms, while a high-level simulation of a full-scale system (64K CMPs) indicates a mean application-loading time of ∼20ms (for a 100KB application), which is virtually independent of the size of the system. We verified the CMP configuration process with hardware-level Verilog simulation.**

I. Introduction

Flexible and efficient boot loading of distributed applications is an essential support process for the SpiNNaker multi-CMP massively parallel system organized over a homogenous communication fabric. The system must somehow break symmetry, assign and load memory resources, configure communications, and start up the processors, while balancing concurrency and resource contention for maximum efficiency. Where previous solutions [4][5] have typically been using sideband communications or dedicated preconfigured resources, SpiNNaker confronts the challenge of configuring an isotropic undifferentiated parallel processing system head-on.

One approach would be to make no assumptions about the application and consider it as a problem in general-purpose computing, leading to a set of standardised, generic configuration techniques. However, since numerous studies indicate that parallel processing works best with specific applications having inherent parallelism, it seems reasonable to design parallel systems around a target application, whose boot process could be correspondingly specialised. SpiNNaker is a Chip Multiprocessor (CMP) for massively parallel spiking neural network applications. Simulating large, biologically realistic neural networks is an excellent candidate application for distributed processing systems: indeed, the consensus in the modelling community is that it may be necessary to use dedicated hardware with architectures more closely similar to the biology for large-scale neural modelling within realistic resource limitations [7]. It is efficient to simulate a spiking neural network as an event-driven real-time application [8], a model quite different from typical parallel applications and more akin to embedded applications [11]. A system for neural network simulation will be, correspondingly, architecturally different from parallel systems designed mostly for general-purpose computing. Dedicated parallel systems such as SpiNNaker mostly adopt event-driven models of computation and boot-time configuration considerations that can make fewer assumptions about the initial state of the system than "conventional" parallel multiprocessor systems.

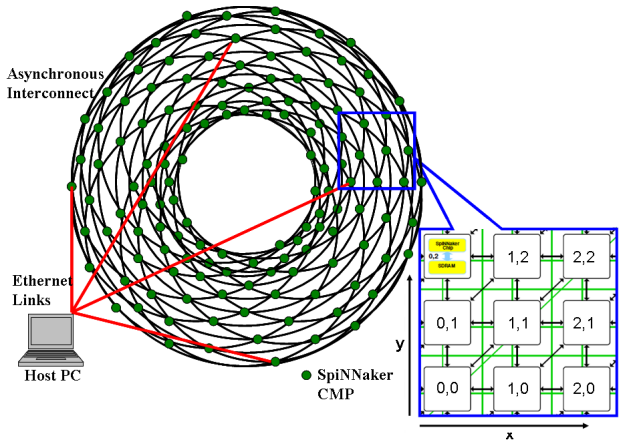

Fig. 1. Multi-CMP SpiNNaker System forming a 2D Toroidal Network.

SpiNNaker provides no sideband communication channel for boot processes: the system boot must use the same communications fabric as the application. All processors on the chip are identical; there is no dedicated processor hard-wired

or preconfigured to run the boot process. The task, therefore, is as follows: It is necessary to configure a symmetric massively parallel system using only the resources available at run time, even though the functionality of these resources themselves depends upon having been configured. The configuration process must do this efficiently and without contention, even though individual processors have only local state information available, i.e. the system can use no global state information to configure itself. We have developed an efficient method for configuring the SpiNNaker system, based on the resources available at run time. The process randomly selects one processor per chip as a "monitor" processor (MProc) and likewise a reference chip in the multi-CMP SpiNNaker system. This breaks the symmetry of its homogenous network, having no starting point and connecting identical CMP's (Fig. 1). We then use the MProc to complete the configuration, broadcasting packets over the network to distribute the neural network configuraton from a central source point to all the chips. The method demonstrates a useful way to leverage inherent asynchronicities in both device and application to achieve a fast boot process in an application-specific parallel system.

II. Real-time NN Simulation Model

A. Biological Information Processing Model

SpiNNaker's target application is the simulation of biologically realistic neural networks. Neural networks are characteristically massively parallel and highly interconnected, a virtual "match fit" for a parallel distributed processing system [1]. Several key properties of "real" neural networks therefore drive the design of the SpiNNaker application-specific architecture. Neurons communicate through spikes: short-duration impulses [1]. It is usual to abstract the spike to an instantaneous pulse, or event, triggered when the neuron reaches a certain threshold value [12]. An actual spike has a duration of about 1 ms, and if we use the point-event abstraction, an update time resolution of 1 ms per neuron is adequate [14]. The spiking dynamics involves a refractory period of the order of ms immediately after a spike during which the neuron will not spike again, setting an upper bound on spike rates of about 100/s, average spiking rates being of the order of 10/s. Neurons typically have high fan-in: $\sim$1000-100K inputs per neuron, but sparse activity: $\sim$0.01-1% active inputs for a given neuron. Such a neuron might then expect 100 events/s in normal operation, with maximum event rates of perhaps 10,000 events/s for extremely active populations. Such event statistics make it practicable to use a "wake-on-event" processing model within SpiNNaker, where individual processors remain asleep until activated by the arrival of an event - a spike.

B. Hardware Support

The SpiNNaker CMP (Fig. 2) is a System-on-Chip (SoC) architecture with multiple (20) processing nodes (processing core and supporting peripherals, such as the Interrupt controller, Timer, DMA Controller and Communication Controller) connected through an asynchronous Network-on-Chip

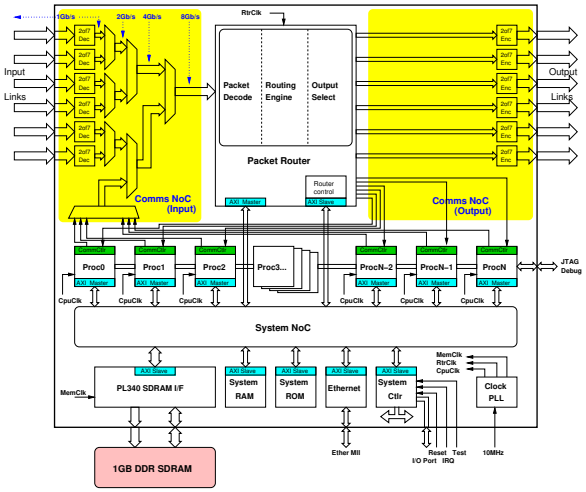

Fig. 2. SpiNNaker CMP.

(NoC). The system is physically configured as a triangularly-connected toroidal mesh (Fig. 1) of CMP's. While the processing cores are general-purpose ARM968S-E's, SpiNNaker's design is optimised for running the spiking neural network models in real time - an Application Targetted Integrated Circuit (ATIC) [2]. SpiNNaker can implement almost any arbitrary spiking neural simulation model within a tradeoff enveloped between model computational complexity, the number of neurons simulated, and real-time update performance.

The inter-CMP fabric (the Communication Network) itself uses a packet-based protocol with three different (40-72bits) packet types: neural multicast packets (MC), point-to-point packets (P2P), and nearest-neighbour packets (NN). The P2P packet is used for system-level management, while the NN packet communicates only among the nearest neighbour chips and is used for chip-level diagnostics and configuration. A chip can send an NN packet to any of its neighbours, broadcast to all neighbours, or use it to "peek" and "poke" the neighbouring chips' resources. The NoC uses a configurable on-chip router to support neural networks with arbitrary connectivity [10]. SpiNNaker has a hierarchical memory system: each processor has only a small amount of local memory (32KB instruction and 64KB data tightly-coupled memory (TCM)); supplementing this is a high-speed on-chip SRAM, shared among on-chip processors. The bulk of memory lies off-chip in a 128MB SDRAM (one per chip), accessed through a proprietary DMA controller over a separate NoC (System NoC). There is one further communication resource for off-system communications: an on-chip Ethernet interface. Each chip has such an interface; in a real system, however, only one or at most a few CMPs actively communicate to the external user interface device, usually a normal PC called the Host PC.

C. The SpiNNaker Execution Model

SpiNNaker's processing and communications are event-driven. Within the system, a set of events makes it possible

to provide fast interrupt-driven hardware support for neural processing while minimising the detailed low-level knowledge the user needs to run a given neural model. An on-board configurable vectored interrupt controller provides hardware support to prioritise interrupts. Three interrupt sources within each processor: a DMA controller, a timer and a communications controller, are central to the model. The timer triggers a "millisecond" interrupt, indicating neuron update timing, while the DMA generates "DMA-complete" interrupt upon completion of a background synaptic data transfer to/from SDRAM. The communications controller issues "packet-received" interrupt upon receipt of a spike.

SpiNNaker implementation decisions impose a few important constraints, critical to the software model design. First, all dynamically updated model parameters must fit into the small 64K Data TCM local memory. Second, since neural updates are timer interrupt driven, any given input update along with the generation of spike events for a given number of neurons must finish before the next timer interrupt (nominally occurring at 1 ms intervals). In [8] we show a way to meet these constraints conveniently while simulating 1000 neurons per processor. With 1000 or more connections per neuron, there are $\sim 10^6$ synapses, far too large to fit into the local memory. We store synaptic information (weight and axonal delay) associated with each connection in the SDRAM, swapping it into local memory as needed using DMA. In [13] we demonstrate that by using fast DMA-based memory swapping over the NoC we can achieve concurrent global memory utilisation, yet make synaptic data appear virtually local to the processor. Axonal delay is a function of spike transmission speed in the model network and is of the order of a few ms [6], however, the communications network transfers these spikes only in a few ns. The spikes received in ∼ns are deferred until the relevant (∼ms) Timer interrupt to ensure a real biological time behaviour from the system. At the end of each interrupt handler the code puts the processing core into sleep mode to conserve power as shown in Fig. 3.

D. Boot-Time Resource Availability

Several features of the design are important in considering the boot process. Since most on- and off-chip processes are event-driven, there is no global inter-process synchronisation and the boot process cannot rely on fixed timing relationships between processors. While each SpiNNaker chip has a large memory resource in the SDRAM, it is not accessible until it has been configured. During application execution, DMA operations effectively "hide" the non-local nature of the SDRAM, but at boot time, this is not the case - and in fact the SDRAM represents another uninitialized, unmapped memory resource. Similarly, the router is blank at start-up and thus the boot process can rely only on the default routing mechanisms to configure the system, while at the same time needing to load the routing tables. The communications network can use only NN packets at boot time, since P2P and MC packets require configured routing tables. The boot configuration process must use the Ethernet interface as an entry point to the system

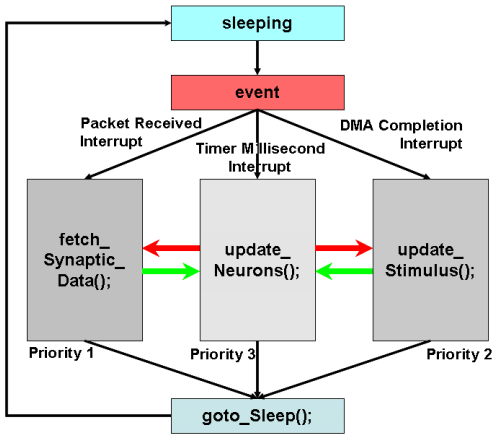

Fig. 3. SpiNNaker Event-driven Neural Application Model.

network, but it does not provide any visibility to the system beyond the chip(s) directly connected to the Host PC using Ethernet ports.

III. System Configuration

A. Configuration Requirements

Particularly at boot time, but also at run-time, SpiNNaker appears as a generic processor resource: a neural "blank slate". To enable it successfully to load and run a given neural network, there are, as a result, several key system configuration considerations.

- For chip-level management, we require one processor out of the 20 on-board to be the MProc. The MProc has 3 roles: pre-boot chip-level configuration and testing, chip-level fault handling, and supporting system-level management. It must perform these management tasks without disrupting the application processors running the neural application. SpiNNaker chips, however, do not have a dedicated MProc, therefore, the boot process needs to select a processor per chip to perform this job.
- To route a packet (P2P or MC) to its destination, the chips need unique addresses. SpiNNaker chips are identical with no hardwired addresses, while the SpiNNaker system is organized as a symmetric toroid with no starting point. Before any application can run on the SpiNNaker system, we need to configure the chips with unique addresses.
- To conserve the CMP area, the Boot ROM size has been kept to a minimum, just sufficient to support initial testing and device initialization. The remaining CMP- and system-level configuration must be performed from outside the system for better flexibility and fault-tolerance.
- SpiNNaker CMPs are not pre-configured to simulate a particular neural dynamic model; they can simulate arbitrary spiking neural models. This means the user initially configures the application outside the system, then loads it. Hence, we need a detailed methodology to

load the neural application with associated data to each chip in an efficient and scalable way.

- A neural network is simply a configuration, not in itself a running application. The boot process needs to configure on-chip routers to conform to the neural network. It is, however, done once and then the target application can be run many times without reloading the system and it is not necessary to consider this time as a function of the time to run a given application - the neural network is not a terminating "program".
- SpiNNaker needs to be attached to a Host PC to load the neural application and interact with the user. Typically, the Host PC would be a normal PC, necessitating some way of connecting SpiNNaker to it. Every chip has an Ethernet interface, but given that only one or at most a few chips would connect to the Host, there must be a protocol bridging the Ethernet communication (between the Host PC and Host-connected chips) and packet-based communication (among the CMPs).
- Once the application is running, we need to interact with the system to examine the state of hardware devices and the application running on the chips. A common communication language using small packets to interact with each chip's MProc is needed.

B. Configuration Process

The SpiNNaker system configuration happens in two phases. In the first phase, the processors run the Boot ROM code in batch mode to test/initialize core- and chip-peripherals independently. In the second phase, the configuration process employs the SpiNNaker event-driven model to configure the whole system from the Host PC.

1) Phase I - Chip Level Configuration: Each SpiNNaker CMP must perform basic power-on testing and initialization based on basic boot instructions in the ROM. At this point, all processors run at very low frequency (10MHz) to save power. After initial testing, the extended boot-up code is copied to the local TCM of each processor for faster boot time. Each processor tests/initializes its local peripherals. Healthy processors compete to access the System Controller (a shared chip component designed for chip-level management) through the System NoC arbiter and the first processor to access the System Controller is selected as the MProc. The System Controller writes the ID of the selected MProc to one of its registers, which enables other processors to identify the MProc. All processors inform the System Controller of their state. At this stage all processors, except the MProc, go into sleep mode. The MProc switches the chip clock to the fast running frequency (200MHz), then performs chip-level testing and initialization of the chip resources. It writes the state of the chip resources to the System Controller for later reporting to the Host PC. The MProc on each chip tests whether a PHY (Ethernet Physical Layer Module) is present. If a PHY is attached to the chip, it may be the one (or one of those) connected to the Host PC. The MProc initializes the Ethernet Interface to start receiving frames. The MProc, at this stage, can reset or disable a faulty application processor to restore or isolate it as part of chip-level recovery. The MProc configures the Interrupt Service Routines (ISRs) to handle packets (and Ethernet frames if Ethernet is enabled) to support the event-driven system-level configuration process. The MProc initiates an event-driven system-level configuration process among the SpiNNaker CMPs by broadcasting a "Hello" NN packet to all its six neighbours, before going to sleep, putting the chip in wait-for-event (interrupt) mode.

2) Phase II - System Level Configuration: In this phase, each chip's MProc runs the configuration process as an event-driven application. The MProc's Communications Controller generates a packet-received event when a message arrives from a neighbouring chip, while at the chip(s) connected to the Host PC, the Ethernet Interface issues a frame-received event to the MProc. Each interrupt triggers the relevant ISR to perform related configuration tasks before putting the MProc to sleep again. The MProc on the host-connected chip translates between the two protocols, i.e. converting the Ethernet frames it gets from the Host PC to packet-based messages, and forwarding them to other chips.

From Phase I, each chip should receive a Hello message from all its neighbours within a certain time. If a given link times out, the MProc activates a "neighbour diagnostic" routine which tries to diagnose the fault. If a chip is non-responsive, one of its six neighbouring chips (with live links to the dead chip) is selected as a "nurse chip" to diagnose the fault through an automated process. The diagnostic algorithm works as an application driven by the packet-received-event. The NN packet is used to "peek" and "poke" the dead chip's resources to diagnose the problem. If the dead chip's MProc is in problem, the nurse chip will reset the dead chip's processors to reactivate the MProc selection process. The System Controller will ensure that the same processor is not selected as the MProc again. If, however, the Boot ROM malfunctions, the nurse chip will use one of the fault-recovery features of the chip, mapping the Boot ROM's address to the System RAM after loading into it the boot code from its own Boot ROM. It will then reset all the processors to restart the chip-level boot-up from the remapped location. In case the nurse chip can not determine a viable recovery solution, it resets the whole chip in an attempt to recover from a transient problem. If nothing works, the nurse chip reports the matter to the Host PC which can run an interactive recovery process with the help of the nurse chip MProc to recover the dead chip or isolate it by disabling all its processors.

Following completion of any neighbour-diagnostic process, the chips start executing interactive system-level configuration. This, the main component of system boot-up, is driven by frame-received- and packet-received-events. The Host-connected chip(s) initiate this process by sending a "Hello" frame to the Host PC, signalling that the system has completed its Phase I process. Thereafter, the Host PC nominates a reference chip to be at the origin address (0, 0) and notifies it of the number of chips in the system. The reference chip then broadcasts its address along with the size of the system.

Each neighbouring chip computes its relative address by size-modulo addition and passes on its own address to the next neighbouring chips. This process continues outward to cover the whole toroidal system, breaking its symmetry by assigning each chip a unique address in the 2D SpiNNaker toroidal mesh. Upon establishing its location in the system, the MProc on each chip configures the P2P routing table with a default pattern based on the logical location of the chips (the Host PC can later modify these tables according to the system-level configuration). The reference (0, 0) chip accumulates chips' status reports using P2P packets and sends them to the Host PC using Ethernet frame(s). The Host PC loads instructions to support the remaining configuration and neural network load process to the chips using a flood-fill mechanism as explained in Sec. IV. The Host PC configures each chip's multicast routing tables as per the mapping and connectivity defined by the underlying neural network being simulated.

A running application can also use the Host-system interactive communication either to distribute stimuli and accumulate responses, or to diagnose/debug the state of the hardware or application.

C. Fault-tolerance

The SpiNNaker hardware supports error detection and handling for several classes of chip component faults with the help of its configuration and management software. The MProc in each chip is responsible for dealing with such contingencies while the other processors continue to run the application. The SpiNNaker system provides redundancy of resources at each level of its design to minimise single points of failure. Each chip's System Controller maintains a continuously updated state of all its processors and shared chip resources. Additionally, most chip components generate interrupts at the MProc to activate a relevant event-driven recovery routine for common faults. The processors, particularly the MProc, are loaded with fault-recovery routines to handle most of these exceptions. One of the main objectives of this research is to keep the configuration process as fault tolerant as possible in order to support real-time applications on SpiNNaker. Dynamic selection of the MProc, configuration of redundant Host-system links, dynamic chip address allocation, run-time configuration of routing tables, System RAM remapping for a dead Boot ROM, and local chip- and system-level recovery are some features that make the process fault-resilient. The application loading process itself includes some aspects of fault-tolerance as explained in Sec. IV

IV. Flood-fill Process

Besides the code to simulate neural dynamics, a typical neural application includes relevant data such as the neurons' state, their synaptic states and the neural network mapping/connectivity information. We need an efficient way to load the application, along with a utility functions library to support chip- and system-level management, from the Host PC to each CMP in a multi-CMP SpiNNaker system in a minimum possible time.

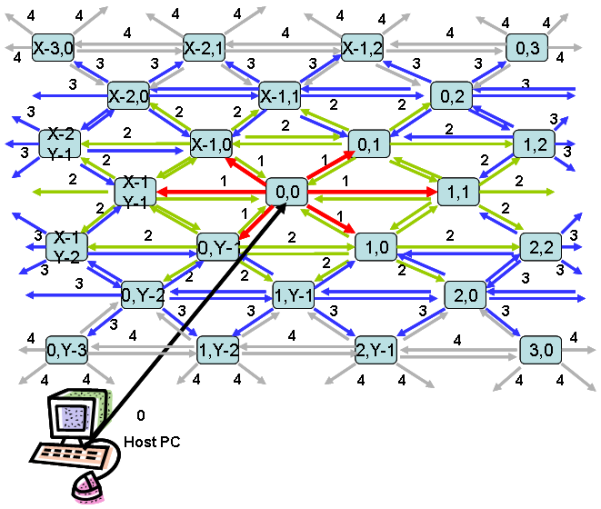

Fig. 4. SpiNNaker Flood-fill Application-loading Process.

An efficient and fault-tolerant "flood-fill" mechanism loads the application and data into the chips as a real-time event-driven process. As part of this process, the Host PC loads the data to the Host-connected chip(s) one data block (1K-16K) at a time using Ethernet frames. The Host-connected chip performs a checksum on the block before transmitting it forward, or requests the Host PC to resend the block. The Host-connected chip informs downstream chips through an NN packet about the block size and its location in the SpiNNaker CMP address space. The data block is transmitted one 32-bit word at a time using NN packets over the SpiNNaker Communications Network. Receiving chips store the data and forward it to their next neighbours. A pipelined "wave" of data thus flows from the Host-connected chips(s) to the whole toroidal system. To achieve meaningful communication control among neighbouring chips, we have devised an instruction set for the NN packets. The protocol includes instructions to serialize the data, control the flow of data, request missing bits of data, and various other control functions. The process ensures that each chip receives every packet at least twice from different directions to ensure successful delivery of data in the event of blocked links. The routing key of the NN packet contains the physical address of the word, identifying the packet and helping in serialization and duplication control. If a word has already been received, it will neither be stored in the memory nor be transmitted further. Missing words can be requested from neighbours at the end of each data block's transmission. The last packet of the data block contains the block-level checksum. If the block passes the error detection test, the receiving chip loads it into the specified location in the memory address space. At the end of the flood-fill process, the Host PC requests the state of each chip along with blocks received. At this stage, the chips can request missing blocks from each other or the Host PC. A dead chip recovering from its faulty state, as a result of the neighbour-diagnostic process,

can acquire the application and data from its neighbours.

V. Experimental Results

We have developed a SystemC system-level model of a single- and multi-CMP SpiNNaker system. The model uses a cycle-accurate ARM968 instruction set simulator (ISS) from ARM SoC Designer together with models for other ARM components such as the Interrupt Controller, SDRAM Controller, Watchdog Timer, AHB, APB bus and memories. All the component models are cycle-accurate and run as part of ARM SoC Simulator. SystemC models for in-house designed components [9] have been added to make it a complete system model. Due to the combination of synchronous and asynchronous parts, the model as a whole exhibits cycle-approximate behaviour as expected from SystemC Transaction Level Modelling (TLM) [3]. The system-level model for SpiNNaker simulates three processing cores for simplicity of simulation which enables the running of a large system simulation efficiently on a host PC (see Table I column 4 for the simulation performance on an Intel Core2 duo 1.6 GHz, 2GB RAM running WindowsXP). The model allows instruction- and cycle-level debugging of the hardware system and the application together, as one package. The simulation results have been verified with a Verilog top-level simulation of the system. The configuration process proposed in this paper has been implemented using ARM Realview Development Studio to generate a loadable Boot ROM binary image. Table I shows the chip-level boot-up time as a number of ARM968 CPU cycles (200MHz). The boot-up time does not depend on the number of chips in the SpiNNaker system since it runs concurrently on all the CMPs. Similarly, it does not depend on the number of CPUs in each CMP as the boot code is loaded to the local memory of each processing core before its execution. We tested an event-driven spiking neural network application [8] developed for the SpiNNaker multi-CMP system after configuring the CMPs as per the outlined configuration process. These results provide satisfactory verification of the design and functionality of the SpiNNaker system.

We also developed a high-level simulator for the SpiNNaker Communication Network to test the application-load process. We could simulate the SpiNNaker multi-CMP system to its full scale (64K CMPs) using this simulation. The communication latency in this model is comparable with the SystemC model as the network timings were those acquired from SystemC cycle-accurate simulation. We evaluated the flood-fill process using the following distribution algorithms:

TABLE I
SpiNNaker CMP configuration Time

CMPs	Procs. per CMP	CPU Cycles	Sim. Time (sec)
1	1	129686	6.00
1	3	129706	8.28
5	3	129706	41.60
9	3	129706	77.86

- broadcast: each CMP's MProc uses the NN broadcast mechanism to send a copy of each packet to all neighbours using only 1 router cycle.
- 2msg: the MProc on each CMP sends messages to its neighbouring chips in the forward direction along X- and Y-axis in a 2D toroidal configuration of SpiNNaker multi-CMP system. The router takes two router cycles to send these packets.
- 3msg: MProc sends messages to three neighbours in the forward direction (along X-axis, Y-axis and the diagonal). We require three router cycles to send three packets.
- 5msg: a node sends a copy of the packet to all neighbours, except the one from which the packet was received. This requires 5 router cycles.
- rndXX: like the 2msg but adds (XX%) probability to send packets to more than two neighbours. We send packets to minimum 2 and maximum 5 neighbours, the decision is controlled by a random probability of 25% (rnd25), 50% (rnd50) and 75% (rnd75) for avoiding a deterministic congestion over the network.

Besides testing the algorithm with all inter-CMP links intact, we experimented with the following link failure models to evaluate fault-tolerance of our proposed flood-fill process:

- vertical: all the links along X-axis are disabled, leading to a network partially split in vertical columns. In this, and the following 2 models, diagonal links remain intact, so the network is not completely split
- horizontal: all the links along Y-axis are disabled, leading to a network partially split in horizontal row.
- cross: the union of the above two, linking the chips only through its diagonal links.
- random: a random set of links failure. We tested the system with various number of links failure from 1K to 64K (total 64K*6=384K links), with a uniformly distribution at various locations.

We tested the system using 1, 2 and 4 Ethernet connections to the Host PC from the CMPs located at (0,0), $(X/2,Y/2)$,$(X/2,0)$ and $(0,Y/2)$ where X and Y are the number of CMPs along X- and Y-axis respectively in the SpiNNaker 2D toroidal configuration (Fig. 1). Finally, we tested different network sizes, all of them square, ranging from 32x32 to 256x256.

Fig. 5 shows our results for error-free configurations with various application sizes to be loaded to the SpiNNaker system. The application loading time depends linearly on the size of data and approximately 20ms is needed to load an application equivallent to the size of a processing core's local memory (100KB). Fig. 6 shows the impact of system size on the performance of various application loading mechanisms, while Fig. 7 shows the impact of the number of Ethernet connections. The results show that the application loading time is essentially independent of both the number of Ethernet connections and system size in a large-scale multi-CMP system. The only relevant factors are the data size and the distribution policies. This is because of the perfect pipelining of the

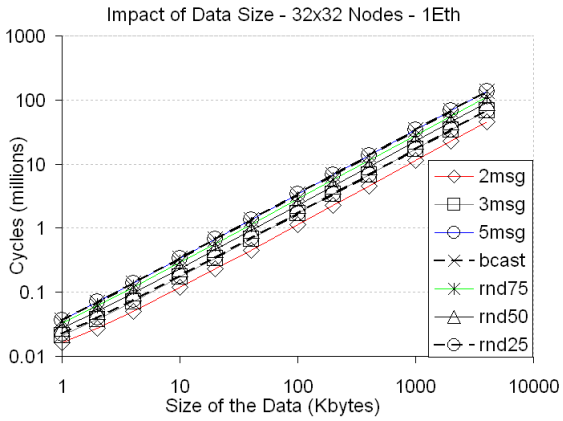

Fig. 5. Application Loading on SpiNNaker Multi-CMP System.

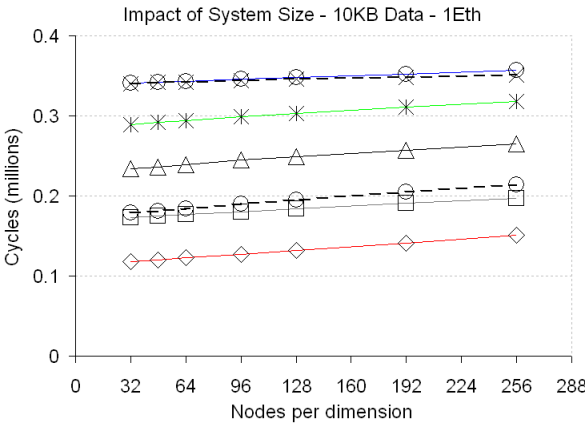

Fig. 6. Application Loading on SpiNNaker Multi-CMP System.

packets in the flood-fill process. Each chip passes packets on to its next neighbours, and in a 100KB sample application the number of hops to reach the farthest point from the origin in the (256x256=64K chips) system is negligible compared to the number of (25K) packets. As a result, the process rapidly transfers application and data across the whole network. The flood-fill process is virtually independent of the number of Ethernet links to the Host, exhibiting negligible performance gain with 4 Ethernet links as compared to 1. There is, however, a considerable performance gain using the selective forward multicast flood-fill process (2msg or 3msg), since this relieves congestion. We found that the 2msg mechanism is the fastest, however, it does not ensure delivery of a packet at least twice to each CMP. The mechanism is not fault-tolerant as a broken link in the start of flood-fill process may deny all the chips in that direction from the application. It is necessary to ensure sufficient redundancy of transmitted packets so that a chip with blocked links should still get the ones from duplicate link(s). Broadcast and 5msg techniques are the most fault-tolerant, however, these have the worst performance due to network congestion caused by injecting too many packets. There is, therefore, a need to maintain a balance between performance and redundancy (fault-tolerance) which is achieved with 3msg and rnd25 mechanism.

Figs. 8 and 9 show the effect of the number of Ethernet connections on the application loading process in the presence of faulty inter-CMP links as explained above. Though connecting the Host PC at more than one point to the SpiNNaker system does not improve the application loading time in a large-scale system, it does improve the fault-tolerance of the process as a chip can receive a packet from various directions. It is particularly important if the network is split in various regions due to link failures. Here, again, the broadcast mechanism proves to be the the most robust by losing no packets in any failure setting, while the 3msg provides reasonably good fault-tolerance as it is not affected by horizontal and vertical failed links and random links failures up to 8K (a system degradation of about 2%).

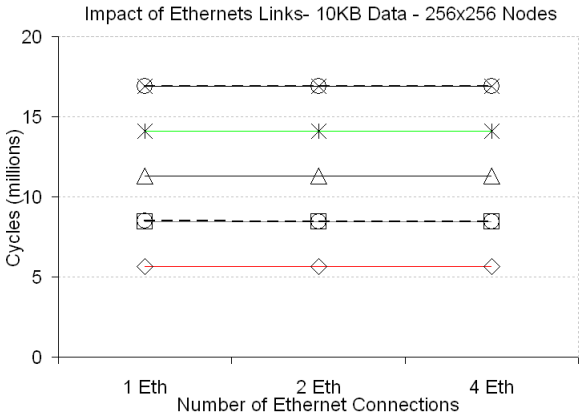

Fig. 7. Application Loading on SpiNNaker Multi-CMP System.

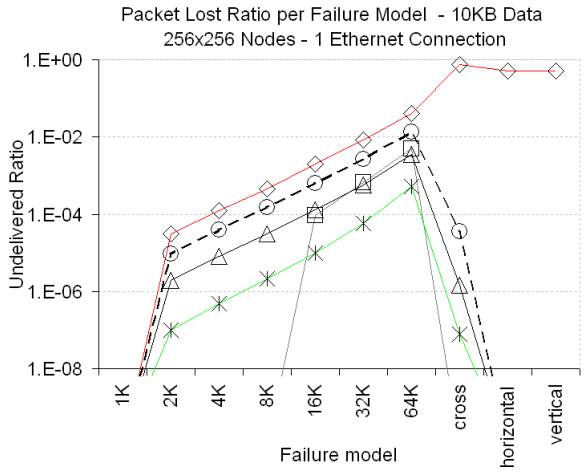

Fig. 8. Application Loading on SpiNNaker Multi-CMP System.

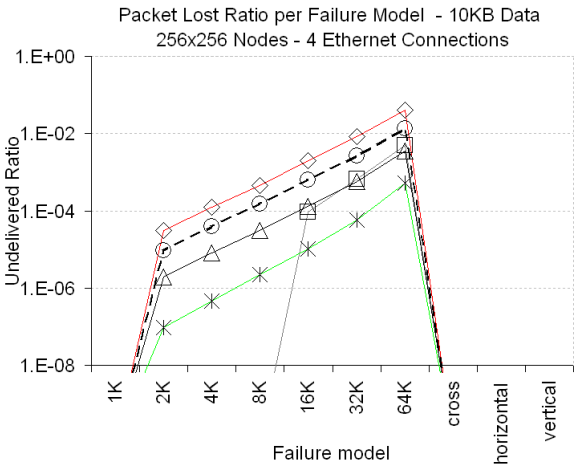

Fig. 9. Application Loading on SpiNNaker Multi-CMP System.

VI. Conclusions

We have developed an infrastructure for configuring user-defined applications on a chip multiprocessor optimised for general-purpose neural network simulation with the following components:

- An efficient fault-tolerant configuration mechanism.
- An efficient flood-fill application load process.
- A library of device routines that abstracts functionality to the model level.
- A cycle-accurate SystemC model for the SpiNNaker system and a high-level communication simulation.

There remains considerable work on high-level user components. We are currently developing a Host PC user interface that provides access to the low-level tools we have already developed. Ongoing research is investigating methods for dynamic and potentially autonomous system reconfiguration at run time, allowing for neural developmental processes. In addition, further work is necessary in identifying efficient routing and mapping schemes. Work is ongoing on developing the library functions and high-level descriptions to support multiple classes of neural network in a general-purpose library of neural functionality that allows the user to specify a model at a high level and automatically instantiate library files to generate the requisite routines and mappings, a process akin to hardware synthesis.

SpiNNaker represents a fundamentally new architecture for neural networks: an Application-Specific Integrated Circuit (ASIC). As such it realises the reconfigurability and ability to model any neural network of FPGAs, and the scalability and performance of traditional hardwired neural ASICs. We have shown that this design approach makes it possible to develop a system that leverages specific performance gains from application-focussed circuitry without constraining model choice or size. Nonetheless, such a "blank slate" approach requires novel methods for configuration and execution or the modeller may be faced with a chip so general that getting it to do anything useful is a research project in itself. Our research has therefore created an essential infrastructure the modeller needs to make SpiNNaker a useful practical tool for hardware neural modelling. Considered as a general parallel architecture, SpiNNaker offers an alternative to the traditional large-scale parallel machine: instead of developing a completely general-purpose chip and designing the application to match the hardware capabilities, we have designed a chip matched to the needs of a specific application known to be highly parallel and have provided general-purpose software tools to develop applications. This represents a path for parallel processing akin to an embedded system where it is understood at the outset that it is running a definable single application. If parallel processing is most effective with specific parallelisable tasks, it seems more logical to develop task-optimised parallel devices to complement general-purpose uni-processors than to try to create a parallel processor to replace the uni-processor outrightly.

Acknowledgements

The SpiNNaker project is supported by EPSRC grant EP/D07908X/1, ARM Ltd. and Silistix Ltd. S.B. Furber holds a Royal Society-Wolfson Research Merit Award. J. Navaridas is supported by a doctoral grant of the UPV/EHU and by the Ministry of Education and Science (Spain) grant TIN2007-68023-C02-02.

References

[1] P. Dayan and L.F. Abbott. *"Theoretical Neuroscience"*. MIT Press, Cambridge, 2001.

[2] S.B. Furber, S. Temple, and A.D. Brown. "High-Performance Computing for Systems of Spiking Neurons". In *AISB'06 workshop on GC5: Architecture of Brain and Mind*, volume 2, pages 29–36, Bristol, April 2006.

[3] F. Ghenassia. *"Transaction-Level Modeling with Systemc: TLM Concepts and Applications for Embedded Systems"*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] R. A. Haring and R. Bellofatto et al. "Blue Gene/L Compute Chip: Control, Test and Bring up Intrastructure". *IBM Journal of Research and Development*, 49(2/3):289–301, 2005.

[5] Cray Inc. *"Cray XT3 Datasheet"*. Cray Inc., Jan. 2005.

[6] E.M. Izhikevich, J.A. Gally, and G.M. Edelman. "Spike-Timing Dynamics of Neuronal Groups". *Cerebral Cortex*, 14(8):933–944, 2004.

[7] A. Jahnke, T. Schönauer, U. Roth, K. Mohraz, and H. Klar. "Simulation of Spiking Neural Networks on Different Hardware Platforms". In *Proc. 1997 Int'l Conf. Artificial Neural Networks (ICANN 1997)*, pages 1187–1192, 1997.

[8] X. Jin, S.B. Furber, and J.V. Woods. "Efficient Modelling of Spiking Neural Networks on a Scalable Chip Multiprocessor". In *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[9] M. Khan, X. Jin, S. Furber, and L.A. Plana. "System-Level Model for a GALS Massively Parallel Multiprocessor". In *Proc. 19th UK Asynchronous Forum*, pages 9 – 12, London, September 2007.

[10] M.M. Khan, D.R. Lester, L.A. Plana, A. Rast, X. Jin, E. Painkras, and S.B. Furber. "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor". In *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[11] H. Kopetz. *"Real-Time Systems: Design Principles for Distributed Embedded Applications"*. Kluwer Academic Publishers, 1997.

[12] W. Maass and C. M. Bishop (ed). *"Pulsed Neural Networks"*. MIT Press, Cambridge, Massachusetts, 1998.

[13] A.D. Rast, S. Yang, M. Khan, and S.B. Furber. "Virtual Synaptic Interconnect Using an Asynchronous Network-on-Chip". In *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[14] T. P. Trappenberg. *"Fundamentals of Computational Neuroscience"*. Oxford University Press, New York, 2002.