Configuring a Large-Scale GALS System

M.M. Khan*, J. Navaridas†, 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*— The SpiNNaker massively parallel GALS system has been designed to support large-scale simulations of biologically inspired neural networks in real-time. The system is built around the chip-multiprocessor (CMP) technology using low-power ARM processors with an asynchronous network-on-chip (NoC) to support high performance parallel distributed processing. A novel asynchronous event-driven boot-up process efficiently configures the SpiNNaker chips and loads the application using a high-speed flood-fill mechanism to a system consisting of up to a million embedded processors in a robust and scalable way.**

I. Introduction

SpiNNaker is an Application Specific Integrated Circuit (ASIC) architecture designed to provide a hardware platform for large-scale spiking neurons simulations in real-time [1]. A full-scale system contains up to a million processors organized in small CMPs connected by an asynchronous packet switching network (Figure 1). Each processing core contains a local memory of 100KB to enable simulating up to 1000 simple spiking neurons. Each processor is an independently functional unit with dedicated resources such as Timer, Interrupt Controller, Communication Controller and DMA Controller; all synchronised to an AHB bus. The processors share chip-level resources such as System RAM, Boot ROM, System Controller etc. using an efficient asynchronous NoC based on CHAIN [2] architecture as shown in Figure 2. While 1000 neurons can be simulated with the help of local memory in each processor, the synaptic information related to these neurons (1000-10000 synapses per neuron [3]) requires much more memory (minimum 4Bx1000(synapses)x1000(neurons)x20(processors)=80MB) per chip. To meet this requirement an off-chip SDRAM of up to 1GB has been provided with each chip. A specially designed DMA with each processor uses the system NoC's efficient throughput (1Gb/sec) to provide a localized view of this data to the neurons in each processor [4]. The performance we achieve with this interconnect at a much reduced energy consumption was never possible with most synchronous bus architectures.

The neurons communicate with each other by sending spikes. The spike communication in the SpiNNaker system has been supported with the help of small packets travelling over yet another fast asynchronous network called "Communication NoC" that connects all the processing cores to a specially designed on-chip multicast router in each chip [5]. The Communication Controller with each processing core provides a bridge between synchronous AHB communication

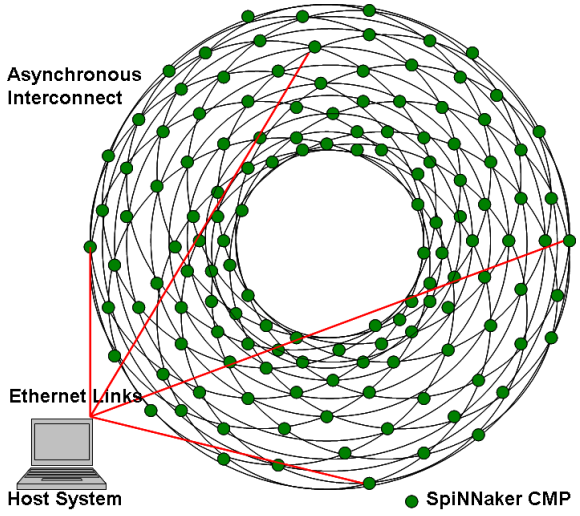

Fig. 1. Multi-chip SpiNNaker CMP.

and packet-switching asynchronous Communication NoC. The router forms a chip-level subnet of the neurons in its 20 processing cores, besides providing a gateway to their communication with the neurons on other chips. The router extends this chip-level subnet of Communication NoC to six neighbouring chips with the help of deadlock free TX/RX interfaces. The global communication network wraps itself around by connecting all the chips in the form of a toroidal mesh as shown in Figure 1, and provides a throughput of 6Gb/s per node [5]. The router is capable of multicasting a packet to any subset of its six neighbouring chips and 20 local processors. Along with the spike carrying multicast packets, the router also supports nearest-neighbour(NN) and point-to-point (P2P) packets that are used for diagnostics/configuration and system-level management purposes. The system is connected to a PC called the 'Host' with the help of the Ethernet connection on the CMP.

The underlying objective is a robust and high-performance architecture at low power consumption [6]. To achieve this aim, SpiNNaker uses low power ARM968 processors and asynchronous NoC along with a power-efficient event-driven application model. The processing cores remain in sleep mode to save energy, woken up by an event such as arrival of a packet or the time to update the neurons' state. The configuration process, uses the same event-driven model to load the application from the Host into the SpiNNaker system

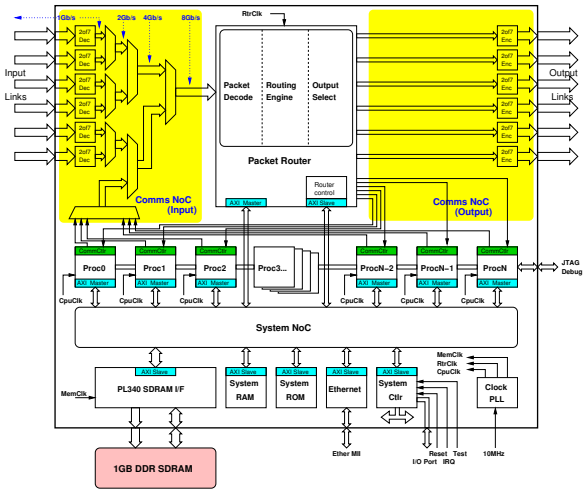

Fig. 2. SpiNNaker CMP.

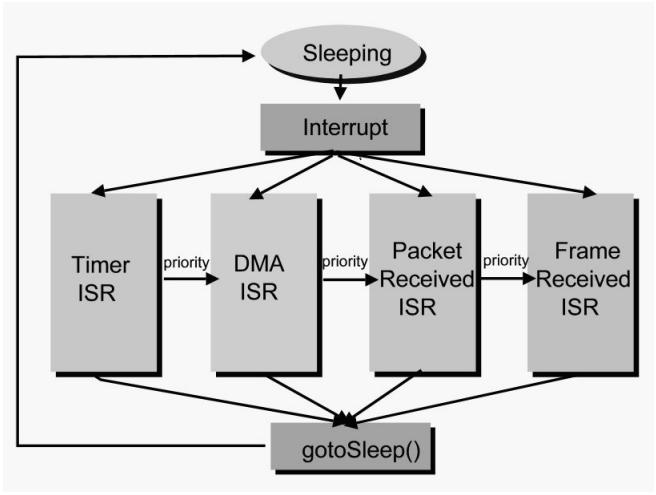

Fig. 3. Asynchronous Event-Driven Model.

using the Ethernet link and the Communication Network. The process provides a scalable and fault-tolerant way to load the application in the whole system very efficiently.

II. Asynchronous Event-driven Model

SpiNNaker uses an asynchronous event-driven model to run the application similar to a real-time embedded application [7]. As per this model, all the processors in the SpiNNaker CMP remain in sleep-mode until an event, such as the arrival of a packet, arrival of a frame, a DMA completion or a timer notification after a certain time interval, wakes them up. These events are provided to the processors as hardware interrupts by the interrupt controller with each processing core. The SpiNNaker uses an ARM Vector Interrupt Controller which can be configured for interrupt priorities. The Interrupt Controller provides the address of an Interrupt Service Routine (ISR) to handle the interrupt for quick branching to the relevant code. The SpiNNaker address space has been distributed in such a way that the software can read the Vector Address Register in the Vector Interrupt Controller in only one CPU cycle. We assign the highest priority to the frame arrival event (on the chips connected to the Host), followed by packet-received event, DMA-completion and the Timer's interrupt, however, it can be re-configured by the user with the help of the application. The event-driven real-time application model has been implemented with the help of ISRs. As part of configuration process, the ISR for a frame-arrival event requests a DMA read operation to bring in the block of data from the Ethernet interface. On DMA-completion event, the state is set for flood-filling the data to the other chips. The next timer interrupt activates a function to start sending the data to the neighbouring chips using NN packets. The packet-received event on neighbouring chips wakes the processor to store the data and pass it onward to other neighbours. The end of every ISR forces the processor to the sleep mode to conserve power as shown in Figure 3. The process has been explained in later sections. The same model, with different function calls from the ISRs, is used for running the neural application [8], once the configuration process is over.

III. Configuration Process

The SpiNNaker has been designed as a general purpose neural network simulator [9]. To achieve this objective, a bare minimum code is loaded into the chips' Boot ROM, just to enable basic power-on self-test and default initialization of chip devices. In order to run any neural simulation model and for better fault-tolerance, the system is configured at the run-time and the application is loaded to each CMP from the Host. The configuration process is divided in two phases. In the first phase, the processors run the Boot ROM code in batch mode to test and initialise processing cores' peripherals independently. One out of the 20 on-chip processors is chosen as the 'monitor' processor with the help of the System NoC arbiter. At this stage all processors, except the monitor processor, go to sleep mode. The monitor processor, which is now managing the chip, switches the processors' clock to the actual (200MHz) frequency, performs detailed chip-level device tests and initializes them to default setting. On the chips connected to the Host through the Ethernet link, the Ethernet connection is configured to establish communication with the Host to start receiving frames. As part of chip-level fault-tolerance mechanism, a chip-level recovery is performed by the monitor processor to restore any faulty chip components using embedded recovery routines. After this the monitor processor also goes to sleep putting the chip in listening mode waiting for an interrupt.

IV. Flood-fill Mechanism

In the second phase, the configuration process runs as an asynchronous event-driven application under the control of the Interrupt Controller. At the chip(s) connected to the Host, the Ethernet Interface generates a 'frame-received' event as an interrupt to the monitor processor. The monitor

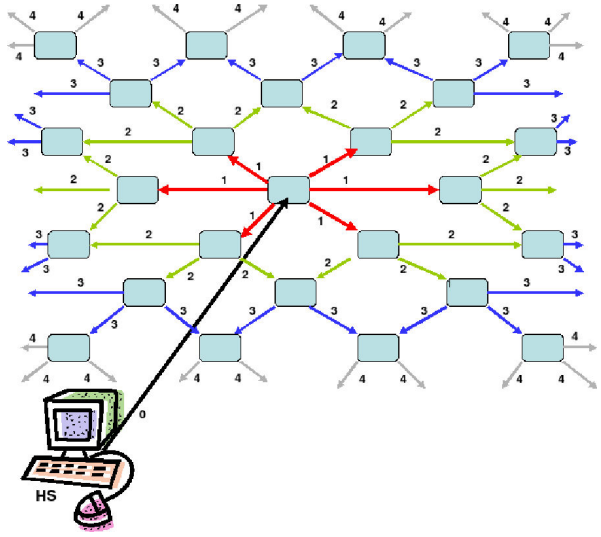

Fig. 4. Selective Forward Flood-fill.

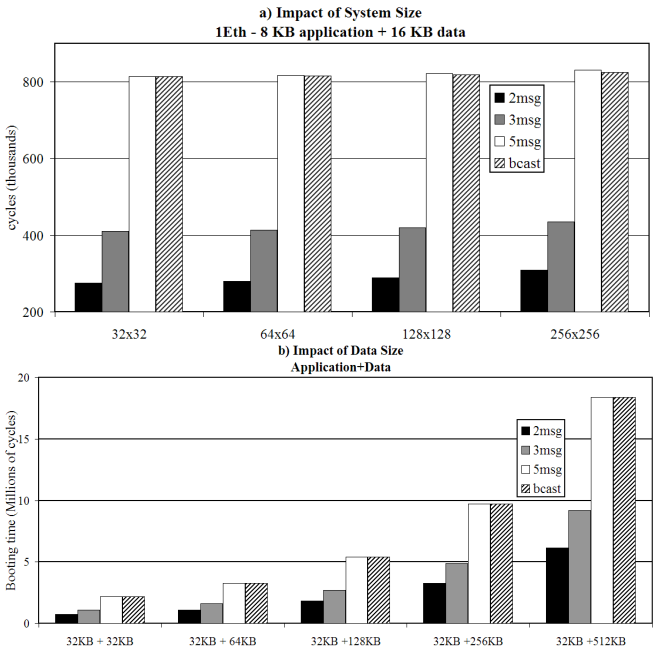

Fig. 5. Flood-Fill Performance Analysis.

processor stores the data into its memory and transmits it to the neighbouring chips as 32-bit words using NN packets. On all chips, the monitor processor's Communication Controller generates a 'packet-received' event when a message arrives from a neighbouring chip. Each interrupt triggers a different ISR to run the code for the related configuration job before putting the monitor processor to sleep again. These two different event-driven processes use two separate protocols. The Monitor on the Host-connected chip translates between the two protocols, converting the Ethernet frames it gets from the Host to packet-based messages, then issuing them either as broadcast or chip-specific NN packets.

At this stage a dead chip in the system is diagnosed by its neighbours and one of the neighbouring chip tries to reactivate it using a diagnostic and a fault-recovery process. This "neighbour's diagnostic" mechanism also runs as an event-driven application with the help of ISRs using NN packets. By using this process a neighbour can inject the boot up code in the System RAM of a chip, remaps the Boot ROM's address to the System RAM and resets the chip's processors to bring it back to life. The Host nominates a reference chip to be the 'origin chip' with address (0, 0) and the chips assign themselves a logical address in 2D (x, y) plane with reference to the origin chip. Each chip configures its P2P table based on the logical location of the chips to perform P2P routing. This can later be reconfigured by the Host according to the system-level configuration. Each chip reports its state to the origin chip using P2P packets. The origin chip accumulates these states and reports the result to the Host. The Host configures the neural mapping and connectivity as per the user's application model according to the chips' state and loads the application to the chips. The Host configures the routing tables for each chip as per the mapping and connectivity defined by the underlying neural network being simulated. The Host instructs the monitor processor to activate the application processors to load the application to their local memories and start running the application. The Host interacts with the system either for stimuli and responses as part of the application or to interrogate the state of the hardware or application.

An efficient and fault-tolerant mechanism has been devised to load the application and data into the chips [10]. During the inter-chip flood-fill process the chips use NN packets to broadcast (or multicast) one 32-bit word of data at a time to their 6 neighbouring chips. Receiving chips store the data and broadcast it to their neighbours onward. A pipelined flow of data thus flows from the origin chip connected to the Host to the whole system. Figure 4 shows one type of multicast mechanisms for flood-filling the data in the SpiNNaker system. As part of the flood-fill mechanism the Host loads the application and data as a data block at a time using Ethernet frames. The origin chip performs a checksum test on the block and transmits it forward if correctly received. The physical address of the word is sent into the routing key of the NN Packet to help serialization and removing duplication. The last NN packet of the data block contains a checksum word. The receiving chips load the data to the specified location in the memory address space. The process ensures that each chip receives every transmitted word at least twice during the flood-fill process, ensuring data delivery to each chip in case of blocked links. At the end of the flood-fill process, the Host 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.

V. Evaluation Work

The boot-up process has been implemented for the ARM968E-S and tested on an instruction- and cycle-accurate

SystemC model for single- and multi-chip system-level model designed with the help of ARM SoC Designer [11]. Phase I of the process i.e. the chip-level configuration takes 129706 CPU cycles (200MHz in actual chip), and is independent of the size of the system and number of processors in each chip as each processor loads the code into its local memory and runs independently. As the SoC Designer does not support real-time delays, the communication delay for the asynchronous part of the system acquired from its HDL simulation was simulated in multiples of CPU cycles i.e. 15ns as 3 CPU cycles at 5ns clock speed. As a large-scale system could not be simulated with this simulation due to underlying PC's resources limitation, the flood-fill process was simulated with the help of yet another high-level simulation based on the timing acquired from SoC Designer simulation for a 9 chips SpiNNaker system-level model. For this purpose we implemented a high-level event-driven simulator that allows the evaluation of different algorithms for flood-fill process. The system model is utterly simplistic but provides a way to scale up to the largest configuration of the SpiNNaker system i.e. with 64K chips. In addition it supports evaluation of the application loading process under scenarios with link failures etc.

In this work we have implemented the flood-fill process by broadcasting NN packet to all neighbours (bcast), sending separate NN packets to the 2 or 3 neighbours in the forward directions (2msg and 3msg), or sending separate NN packets to all neighbours except the one the packet is received from (5msg). We have also tested different sizes of the SpiNNaker network, ranging from 32x32 to 256x256 to load varying sizes of data. The results in Figure 5 show that the application loading time is almost independent of the system size. This is because of the perfect pipelining of the packets in forward direction. Apparently, the only factors affecting the performance are the size of the data and the flood-fill mechanism i.e. broadcast or separate NN packets to the neighbours. It is obvious that the fewer packets the router has to send the better performance in terms of time is obtained. In the case of broadcast mechanism, though the processor sends one packet that is broadcast in one router cycle to all the neighbours, contention at consumption reduces the performance. From the results, the best performance mechanism is to send individual packets to only two neighbours in the forward direction (2msg), however, this policy is not fault-tolerant as a blocked link in one direction may deny all the chips in that direction from that packet. The mechanisms with broadcast or with sending packets to at least three neighbours in the forward direction are more robust as these guarantee duplicate packets to the recipient chips.

VI. Conclusions

The SpiNNaker architecture, using real-time embedded application model over low-power ARM968 processors and asynchronous NoC, provides an energy-efficient yet powerful platform to enable large-scale spiking neural simulations. To keep the architecture universally suitable for all kinds of neural modelling, the system can be configured for the neural application and neural connectivity at the run-time. A novel configuration process has been devised as part of this research to meet this aim. With the help of this process, we can load the neural application and relevant data besides configuring the system to support the neural connectivity as per application setting. The simulation results show that the process is very efficient, scalable and fault-tolerant to support a novel large-scale massively parallel GALS architecture. We are currently working on the user interface at the Host and to provide a library of functions to enable configuring the application by abstracting the details of the SpiNNaker architecture from the user. The interface, once completed, will provide an automated process of configuring the application at the Host before loading into the SpiNNaker system to optimally utilise the design features of the SpiNNaker ASIC architecture.

Acknowledgements

The SpiNNaker project is supported by the Engineering and Physical Sciences Research Council, partly through the Advanced Processor Technologies Portfolio Partnership at the University of Manchester, and also by ARM and Silistix. Steve 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, and by grant IT-242-07 from the Basque Government.

References

[1] S. Furber, S. Temple, and A. 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] L. Plana, J. Bainbridge, S. Furber, S. Salisbury, Y. Shi, and J. Wu, "An On-Chip and Inter-Chip Communications Network for the Spinnaker Massively-Parallel Neural Net Simulator," in *Proc. Second ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2008)*, 2008, pp. 215 – 216.

[3] R. F. Thomspon, *"The Brain - A Neuroscience Primer"*, 2nd ed., G. L. R.C. Atkinson and R. Thomspon, Eds. New York: W.H.Freeman and Company, Worth Publisher, 1997.

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

[5] L. Plana, S. 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*, vol. 24, no. 5, pp. 454–463, Sept.-Oct. 2007.

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

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

[8] X. Jin, S. Furber, and J. 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] S. Furber and S. Temple, "Neural Systems Engineering," *Journal of The Royal Society Interface*, vol. 4, no. 13, pp. 1–14, April 2006.

[10] M. Khan, J. Navaridas, X. Jin, L. Plana, J. Woods, and S. Furber, "Real-Time Application Support for a Novel SoC Architecture," 2008, to appear in Proc. of 4^{th} UK Embedded Forum Southampton UK, September 2008.

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