Social-Insect-Inspired Networking for Autonomous Fault Tolerance

Matthew Rowlings, Andy Tyrrell, Martin Trefzer
Intelligent Systems Group, Department of Electronics, University of York, York, YO31 5DD, UK
Email: mr589@york.ac.uk, andy.tyrrell@york.ac.uk, martin.trefzer@york.ac.uk

***Abstract*—As electronic hardware integration technologies develop there is an increasingly strong shift towards implementing complete systems within a single chip, extending the now established paradigm of System on Chip towards high density many-core systems by employing Networks on Chip (NoCs) to connect the processing elements. This brings many new challenges to fault-tolerant design when applied to embedded applications, but also opportunities for new approaches that can leverage the many-core fabric in ways that traditional system architectures could not exploit. This paper describes such an approach by adopting behavioural aspects of social insects as an inspiration towards autonomous, self-repairing systems. Each router in the NoC is considered as a member of a distributed colony and a simple adaptive controller is responsible for determining the behaviour of each node, relying only on a set of sensory inputs local to each node and small amounts of information shared between neighbours. This provides each node with a small amount of "intelligence" that, for this paper, has been implemented within a many-core hardware system to demonstrate an adaptive routing scheme which provides effective network traffic management through simple and decentralised agent-to-agent communications. The emergent behaviours of the network are then exploited to demonstrate an example of how fault tolerance could be supported within a many-core system without any predefined fault handling strategies. A discussion then follows on how the emergent behaviours of this system can be further inspired by social insect colonies to exhibit other autonomous and adaptive behaviours such as dynamic task allocation.**

I. Introduction

The extremely high logic density of modern VLSI platforms has lead to an adoption of many-core systems for embedded system design, relying on a *Network on Chip (NoC)* [1] [2] to interconnect the processing elements of the many-core array. Whilst the NoC shares many properties with conventional computer networking, the application to embedded systems means that the network should be designed to conform to typical embedded system constraints such as power efficiency, compact resource requirements and effective fault tolerance. Therefore an inclination to simpler networking capabilities is seen with NoCs when compared to conventional networking, a caveat of this is that performance suffers and so a trade-off between node router complexity and performance has to be made. An alternative is to perform offline analysis [3][4][5] and optimisation of the task and network model, however the resulting strategy is generally fixed and so does not support runtime dynamic reconfiguration of the system structure; a key requirement for supporting future many-core system design paradigms such as dynamic task allocation, in field self-repair and autonomous online optimisation[6].

Thus we need a network that can self-organise and self-optimise without the need for offline analysis. To support both good scalability and dynamic network topology reconfiguration, an ideal routing algorithm should therefore not rely on global knowledge of the network layout; indeed if many-core systems do scale into the hundreds and thousands of cores as suggested in [6] then any online analysis will be computationally infeasible within an embedded system. Therefore the network will have to take a decentralised approach to routing, whereby each node in the network is responsible for its own routing behaviour. A simple example of this is the Round Robin algorithm [7]. By servicing each port in turn and only allowing each port to be serviced once in a round, Round Robin provides a decentralised and fair routing strategy that does not rely on any global coordination. Whilst the authors appreciate that Round Robin is a very simple case in a field full of more capable algorithms for specific applications, its simplicity means that not only is it suitable for implementation of a NoC in an embedded system but it also serves as a good baseline to compare the self-organising algorithms proposed in this paper to.

When researchers consider self-optimising systems, many have looked towards Nature for inspiration. Life has provided a host of examples of decentralised self-organising systems at all ranges of abstraction: from the chemical networks used for gene regulation, to the cellular growth and development in multicellular organisms, up to the social networks required for survival of insect (and other) colonies. However when these models are applied to engineering problems we often see significant overhead requirements due to the extra resources required to fit the engineering model to the biological metaphor. The UNITRONICS project for example [8] uses multi-cellular development as an inspiration for building fault tolerant systems but a simple 4x4 hardware multiplier required 40 cells, where each cell requires significant hardware resources to support all of the cell development model; an arguably large amount of unnecessary overhead for a simple circuit that makes scalability across a whole many-core system infeasible.

Therefore when considering inspiration from Nature, it is important to find good links between the metaphor and the target problem. For many-core networking we can break our problem down into a decentralised model with simple communication between neighbouring nodes in addition to local knowledge available at each node. To implement this on chip requires a model as efficient as possible, whilst bearing in mind that although Nature produces efficient solutions they are not guaranteed to be optimal! Considering this, we argue that social insects are a suitable metaphor for many-core networking as their communication structures fit the decentralised model well, simple communications between members result in self-organising behaviours emerging when observed globally at colony level; examples include nest building, self-replication and food scouting, gathering and dispersal. Their typical habitat has resulted in Nature to produce behaviours that are very food (i.e. energy) efficient, fitting the embedded system application case well; indeed successful use of social insect models in other problem areas have succeeded because of the efficient emergent high level behaviour (ant colony optimisation applied to the travelling salesman problem for example [9] [10])

This paper describes a series of experiments exploring the application of social insect inspired network routing schemes to a simulated NoC. First each of the decentralised routing algorithms are described in terms of their biological inspiration and the local sensory inputs available to each node. The experimental setup is then introduced including how these algorithms are mapped to the implementation platform. Section IV then presents a statistical analysis of the performance of each routing scheme on two representative NoC applications. Finally the results presented here inform discussion in Section VI on how the model may be elaborated to exhibit autonomous NoC topology optimisation.

II. Intelligent Many-Core Routing

To investigate which behaviours of the social insects can be applied to hardware systems, biological observations of the social insects are examined. Those of particular interest include [11] wherein the authors argue that a honey bee can be described in terms of a behavioural repertoire of 59 distinct behaviour patterns and [12] which explores several models of agent task allocation and how this is determined at the single organism level to result in an emergent task allocation pattern at colony level. These distributed models have a clear analogy to desired properties of many-core systems; each node in such a system could be modelled with a distinct behavioural repertoire depending on its function and then the repertoire of each node (*or agent*) is exploited at the local level to provide a scalable, distributed system with the desired emergent properties demonstrated by social insect colonies, i.e. scalability, adaptability to new environments and colony fault tolerance. Instead of the more traditional design-space analysis approach to network-on-chip routing, we considered several agent level behaviours that are implemented heterogeneously at each node in the many-core grid. Taking inspiration from [12] we used a simple threshold intelligence model at each node which takes data from sensors local to the node and performs a routing behaviour depending on the input to the sensors and the threshold function applied. In each investigation step different sensors are incorporated into the threshold model; with the aim to capture the following social insect inspired features within the many-core system:

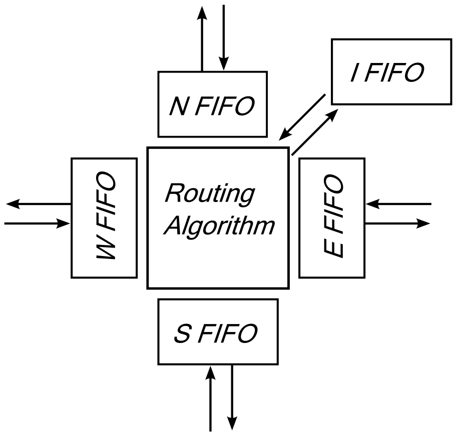

Fig. 1. Five port router design. The internal link represents the application node and the North, East, South and West ports connect to neighbouring routers

1) Simple sense/act behaviour of insects
2) Efficient communication strategies between agents
3) Emergent overall behaviour of the colony

The remainder of this section describes the sensory capabilities investigated, with each capability striving to add more local information to each node with the ambition to improving the effectiveness of the emergent routing behaviour of the many-core system. Each scheme assumes a five port router as illustrated in Figure 1, this is a typical design for a router with an attached application node and neighbours at each of the cardinal points.

A. Round Robin Routing

To provide a suitable baseline to evaluate the proposed algorithms against, a decentralised routing algorithm that does not use any local node knowledge is required. Such a port servicing algorithm is the simple, but well used, *Round Robin* arbitrator[7]. In this routing scheme each port is checked in turn for data. If a port has data to route then a single packet is routed and then the next port in the router is checked for data. This continues serving ports in a circular fashion (in our case N,E,S,W,I and back to N), providing a fair and balanced routing scheme without utilising any of the sensory information available to the node.

B. FIFO Fill Threshold Model

The first investigation architecture looked at the possibility of balancing the network's overall load by balancing the traffic at each node. To achieve this each node was given knowledge of the fill levels of each of its North, East, South, West and Internal FIFO buffers. A simple threshold model then chooses the router port to service based on which node has the most traffic waiting in its buffers, i.e:

$$\text{Port} = \max(\textit{fill}(x)) \qquad \forall x \in \{N, E, S, W, I\}$$

where *fill(x)* is the FIFO fill level of port x

C. Neighbour Hunger Model

The previous models have considered how sensory inputs can be used to decide which port to service, this model however considers the output port that a packet should be routed to. As introduced in Section III, each node has an ordered preference list of output ports that it should route a packet to dependent on its task. This ensures that short paths are taken through the network, but it can also mean that certain nodes are put under lots of pressure in congested scenarios. Sometimes a longer path through the network will result in a more balanced traffic flow, potentially improving throughput over the entire network. To investigate this we have considered the food distribution networks of ant colonies [13]. The nature of task polyethism in the colony means that food must pass from returning foragers at the front of a nest all the way throughout the entire colony to keep all members well fed. From a simplistic point of view this is achieved by individual members taking more food than required for themselves when offered and then sharing it with other members as they pass them in the nest. If the recipient already has a plentiful amount of food then she refuses the offer and the supplier tries other members until her food excess is removed. This self-organising, distributed methodology successfully balances food distribution across the colony; this investigation explores if can we use such a scheme to balance packet distributed across the NoC.

As a node's buffers fill up, their fill is accumulated and compared to a "hunger" threshold, the output of which is shared with each of a node's neighbours. When this threshold is passed then a node is no longer "hungry" and its neighbours will not send it any more packets until it becomes "hungry" again. To achieve this the ordered preference list of output ports is used to select an increasingly less optimal output port should the neighbour on the optimal output port not be hungry, if all neighbours are not hungry then the router does not route any packets until one of them is.

Formally, for each node:

$$\textit{Hungry} = \textit{'Yes'}\ if\ \{\textstyle\sum(\textit{fill}\ x) < \theta\,\} \quad \textit{else 'No'} \qquad \forall x \in \{N, E, S, W, I\}$$

And when servicing a port:

$$\textit{Output Port} = \textit{first}(y)\ \textit{where Hungry}(y) = \textit{'Yes'}$$

where θ is the total FIFO fill level at which a node is no longer hungry and *fill(x)* is the FIFO fill level of port x as with the first model. y represents a list of the the node's N,S,E,W neighbours (which may not exist if a node is located on the edge)

D. Neighbour Hunger for Fault Tolerance

The hunger model introduced above can straightforwardly be exploited to support node-level fault tolerance. If we assume that each node contains a fault detection method this could be used to enforce a "not hungry" state when a fault is detected. Other routers would then avoid sending data via this router and would instead use the hunger model to route packets around the faulty node, then to be ultimately processed by a different node but of the same target task. Aside from the resources required for fault detection in the node, a node-level fault tolerance strategy can be supported by the system without any extra resources required; a good example of the powerful emergent properties that we can anticipate by taking inspiration from the social insects.

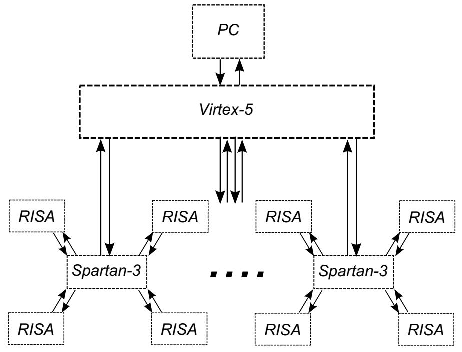

Fig. 2. Many-RISA system layout. The host PC is responsible for generating and managing the experiment setup. The Virtex-5 acts as a mediator, forwarding messages from the host PC to the Spartan-3s and also collating results and status information from the Spartans. The RISAs perform only the routing algorithm under experiment where the Spartan-3s act as the internal port for a cluster of four RISAs, taking the role of an application node by dispatching and accepting packets

III. Experimental Setup

To facilitate characterisation of the behaviours of the presented routing schemes within a hardware implementation, a many-core system was built in a fashion representative to modern NoCs. A total of 36 Reconfigurable Integrated System Array (RISA) [14] integrated circuits were used to fashion a 6x6 many-core system, utilising the cardinal serial ports of each RISA to produce a mesh interconnect. The RISA platform consists of an embedded RISC processor coupled to a small reconfigurable fabric, making the platform ideal for implementing self-optimising structures at node level. Whilst this network of processors is not strictly a Network-on-Chip, the processors and their serial communications are basic enough that they easily map to asynchronous NoC interconnect technologies; indeed it would be possible to implement a NoC of RISAs in a FPGA or ASIC without any redesign of the platform.

The array was constructed by combining nine boards, with each board comprising of four RISA chips with a Xilinx Spartan-3 FPGA in the centre with a serial and GPIO interface to each RISA. This allows the desired router design given in Figure 1 to be created, whereby the Spartan-3 simulates the application node for all four RISAs through each of its serial interfaces and the RISA processor only performs routing operations. The GPIO interface is used to communicate status information to ensure that the routers are not malfunctioning and is used in the "Neighbour Hunger Model" to report the hunger status of a node's neighbours. When the experiment is running the Spartan-3s dispatches packets to their various RISA routers according to the profile generated by the host PC which formulates a packet generation formula from the following constraints:

- Maximum packet rate: the minimum time between successive application packets
- Input data dependency: the minimum number of packets a node must receive before it can send a packet out
- Processing time: the minimum time between receiving a packet(s) and sending a packet out

These constraints allow a diverse range of application node behaviours to be simulated, a key concern for experimenting with realistic application scenarios. The Spartan-3s also receive packets from the RISA routers, whereby they calculate and store the packet traversal time (through the use of a sent timestamp stored in the packet) and also update the inputs to the above packet dispatch constraints.

Overall control of the experiment is managed by a host PC which is responsible for generating the NoC topology, generating the packet creation profiles, starting and stopping the experiment and collecting the results. A mediator between the host PC and the Spartan-3 boards was required, this is in the form of a Virtex-5 board which manages the communication between the Spartan-3 boards, including synchronisation of each board's real-time clock (for packet traversal measurement) and communication of the hungry status of each RISA. The Virtex-5 also forwards commands to and from the host PC and the Spartan-3 boards, as well as providing support for updating of the software and bitstream of the Spartan-3s and updating of the RISA software. This is illustrated in Figure 2.

Implementation of the routing schemes was built around a set of common routing functions working on input data buffers connected to each port. These common functions consisted of:

- Reading a packet from the Spartan-3 serial link
- Writing a packet to the Spartan-3 serial link
- Reading a packet from a {*N, E, S, W*} RISA serial link
- Writing a packet to a {*N, E, S, W*} RISA serial link

For the Round Robin scheme, each of the N, E, S, W ports were checked for valid packets in turn and one packet routed if available before moving on to the next port. The FIFO fill level scheme requires a little more knowledge and this is achieved by keeping a count of how many packets are waiting to be routed at each port. The router then decides which port to service depending on which port has the most data ready to route. These two schemes rely on a pre-defined optimal routing direction for each task, this is generated by the host PC as part of the task profile generation and optimally is based on a simple Manhattan distance metric. The neighbour hunger scheme however extends on this information by utilising a pre-defined *ordered* list of routing directions for each task. By ordering this list by optimality we can choose sub-optimal choices when the chosen routing direction is no longer hungry. The hunger status of each node is communicated via the GPIO to the Spartan-3 and then distributed via the Virtex-5 information network, however in different network schemes this could be also implemented as a dedicated signal or status packet.

To evaluate the hungry scheme under faulty conditions a fault injection scheme must be supported. This is easily achieved at node level in this system via manipulation of the hungry status of a node. A number of nodes are randomly selected to be faulty (depending on how many faults are required) and the Spartan-3 responsible for each faulty node communicates that this node is not hungry and does not dispatch packets to it. This has the effect of forcing all packets to route around this node (desirable in a faulty situation) whilst also slightly reducing traffic local to the node as the node is no longer dispatching new packets, representative of a fail-quiet scenario. This will alter the balance of traffic in the network which our routing scheme should exploit to recover some of the routing performance of the network. The packet generation rates of other nodes of the same task as the faulty node are proportionally increased to ensure a realistic application case, otherwise faulty nodes would actually lower the routing requirements of the network thus this ensures there is always the same amount of traffic in the network despite the failing nodes.

IV. Experimental Results

A series of experiments using the many-core system described in the previous section are presented here. Two application scenarios are considered to explore the strengths and weaknesses of each strategy under different operating conditions. Each experiment records the average time taken for a packet to traverse from its source node to its target node and its packets are continuously dispatched at a predefined rate until 5000 packets have been sent throughout the network. The experiment then waits for all packets to finally reach their target node. This experiment is repeated 100 times with a different randomly generated network topology in each run. This allows a statistical outline of the performance across many variations in network node topology to be measured, capturing the mean performance as well as the worst and best case outliers. This is performed for each of the schemes and for each application scenario, shown in Figures 3 and 4: (1) Round Robin, (2) FIFO Fill Threshold and (3) Neighbour Hunger Model.

A. Experiment Application Scenarios

The application scenarios dictate the traffic flow across the network through definition of the following parameters:

- Number of different tasks
- Ratio of task allocations
- Size of packets generated by each task
- Rate of packets generation by each task
- Number of packets from other tasks that the node has to receive before sending out a packet (causality)

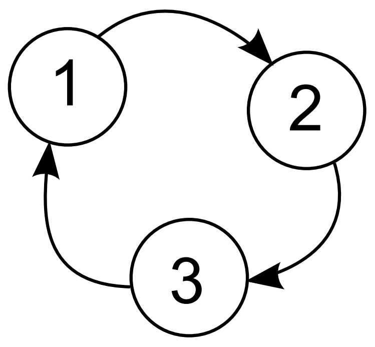

Fig. 3. Application graph for the first scenario. This represents a simple balanced processing application where each node produces data at the same fixed rate which is only consumed by one task. This creates a balanced traffic profile across the network, perturbed only by the network topology.

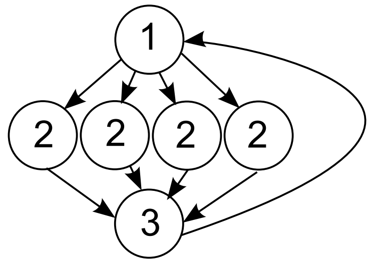

Fig. 4. Application graph for the second scenario. A data pipeline with a parallel stage is represented here whereby there are four times as many task two nodes as task one or three nodes. This can represent a typical many-core streaming application with a stage that is massively parallel, however all data rates are kept the same as in the previous, balanced application graph shown in Figure 3. This scenario effectively increases the load on routing to task three nodes.

These parameters allow many different application graphs to be applied to and tested on the many-core array. For these experiments we shall represent two simple, time-trigged applications where each task dispatches constant sized packets at a constant rate. All that we shall change between the topologies is the ratio of task types. Three different tasks were chosen as this scales equally to the 36-core array and is the smallest number of tasks that can offer the one-to-many-to-one model that scenario two investigates. Figure 3 illustrates the application graph for scenario one. In this scenario there are an equal number of tasks one, two and three distributed randomly across the array (i.e. 12 nodes of each task). Each of these tasks dispatch a new packet of 50B every 5ms, this results in a traffic flow that saturates the network as the minimum traversal time (from one node to its neighbour) for a 50B packet is 4ms on the array. Scenario two imbalances this by employing a 1:4:1 task ratio across the network, as illustrated by Figure 4. This represents a typical many-core application where a specific task is shared amongst many nodes as it may be computationally expensive and can gain from parallelisation or as part of a N-modular redundancy voting scheme.

B. Routing Performance

The average packet traversal times for each task for 100 runs of the experiment for the first task scenario are shown in Figure 5. It is clear from the bias of the box plots towards the minimum traversal time that many packets are sent directly to a neighbour node; this is to be expected to some degree when using only three tasks in the application graph, and each task only has one of the two other tasks as its possible target. With four neighbours, the chances of having the target task as a neighbour is quite high. Additional to this bias we also see a large spread of outliers in the Round Robin case, this is due to Round Robin treating a busy port in exactly the same way as a quiet port i.e. a port with 5 packets queuing would have the same routing priority as a port with 30 packets queuing. This results in packets waiting at busy ports for long periods of time, something which the FIFO fill level monitor manages to balance by allowing the router to clear busy ports first. Some deviation away from the optimal packet latency is still seen as congested routes are not avoided, but it is clearly seen in Figure 5 that the extreme outliers are no longer present. The Hunger scheme improves on this by effectively utilising that little bit of information shared between nodes to avoid hotspots and capably balance the network traffic overall. This has the result of bringing the medians of all tasks closer to their best case and producing a much smaller distribution of the results regardless of the differing network topologies.

Despite the different application profile for scenario two, we would expect a comparable behaviour for each of the schemes. Indeed as can be seen in Figure 6, it is again clear that the Hunger scheme can drastically improve the network performance. The effect of the different task model is also obvious from all the schemes. Packets from task one have a high ratio of task two nodes to finish at and so this distribution is relatively tight. The opposite is seen for packets sent by task two, as many task two nodes have to send to a smaller ratio of task three nodes. Depending on the topology this could result in a task three bottleneck where only a few task three nodes are very busy sinking packets dispatched by all the task two nodes. Due to the fairness of Round Robin we can see that the distribution for task three has a small spread, this is smaller than task one due to “knock-on” effects of the traffic from task two nodes: as the traffic from task two nodes struggle to arrive at their task three destinations, consequently we get a delay for receiving incoming task one packets as well. The FIFO fill strategy sacrifices this task three performance to improve the upper bounds of task two packets, albeit the Hunger model again uses its extra information to far greater effect to eliminate the extreme outliers and reduce the distribution of packet traversal times by ensuring that packets are well distributed across to all task three nodes. The “knock-on” effects for task one nodes are mitigated through the dynamic routing of task two packets to different task three nodes as the hotspot task three nodes become saturated and thus no longer “hungry”. This experiment highlights another power of such decentralised strategies as they exhibit their behaviour across

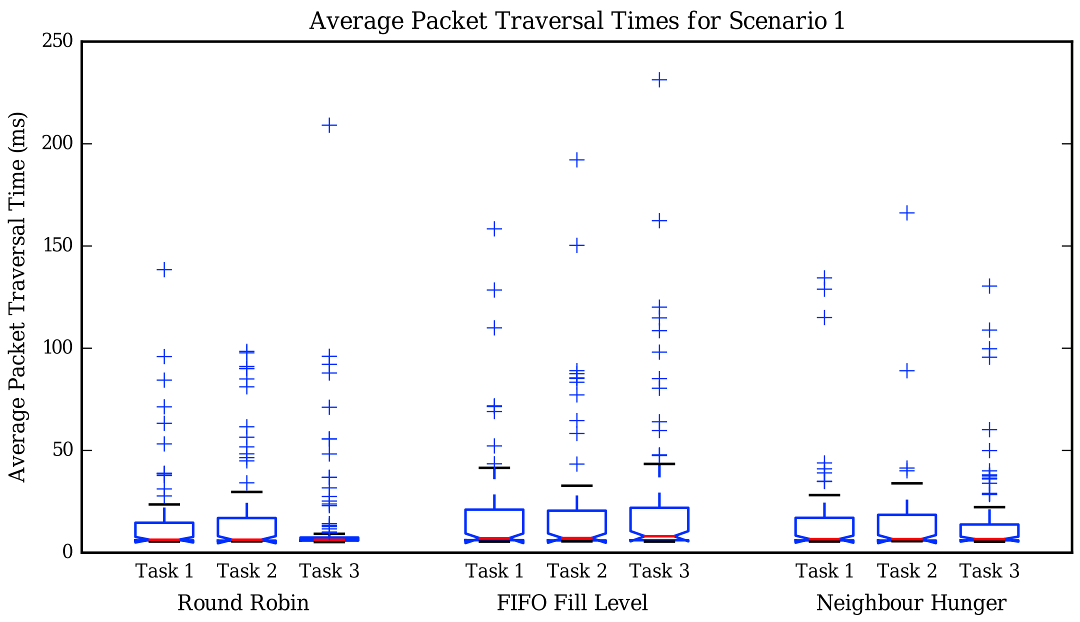

Fig. 5. Average packet traversal times for the first task scenario. As the number of nodes of each task in this application is equal and all tasks share the same packet generation parameters, we expect no significant variation in performance between tasks. The only difference between runs is the randomly generated network topology which, due to the selection of parameters that provide a congested network, can have a severe impact on the network's behaviour. It is clear that the Neighbour Hunger model is far more effective at mitigating the congestion and sub-optimal topologies, with both a smaller range and very few outliers with no extreme values. The skew of all of the distributions towards their lower quartile is due to many packets reaching their destination in an optimal time, with only three tasks it is very likely that a node will have a neighbour as its packet target and so can complete the transfer in a single network hop.

many different operating situations, an important feature as implementation specific attributes such as task graphs tend to vary significantly between applications.

C. *Routing Performance Under Fault Injection*

We can now explore the fault tolerance properties of the Neighbour Hunger model. As discussed in Section II the hungry flag can be exploited to exhibit fault tolerant behaviour and Figure 7 shows the average packet latency for up to five faulty nodes randomly introduced into the system before running the experiment. As the task allocation topology was kept constant we only expect differences in packet latency to be due to where the fault occurs (i.e. at a routing bottleneck or not). It is important to note that no packets are lost in this scheme; Round Robin for example would continue to route packets to faulty nodes despite their defective status.

For one or two faults in the system we see the expected slight increase in packet traversal time, with two faults starting to introduce more outliers into the results. What is then interesting is that the distribution then stays fairly regular despite increasing faults in the system, this is likely due to the emergent fault tolerance of the Hunger scheme exploiting the uniform layout of the nodes i.e. an alternative destination node is not that much further away from the source node than the now faulty node. Indeed when the results of the second scenario are considered in Figure 8 we see the imbalance in the application graph has a drastic effect on packet latency, but it is worth noting that the medians are still scaling quite closely to their lower quartiles. The location of the injected fault is even more significant when it is considered that a task three node could be removed from the system, causing even more strain on task three resources that the Hunger scheme can only mitigate so far. Indeed this is a good example of where dynamic task allocation could be used in addition in a fault tolerant system as a further means to "intelligent" congestion relief at the node level.

V. Conclusions

The results presented have shown that the Social Insects can provide effective inspiration for self-optimisation of many-core systems in a fully decentralised fashion. An in-hardware simulation of two different application scenarios yielded a very similar emergent behavioural pattern, suggesting that this approach can be applied to a wide range of applications without requiring any NoC topology pre-analysis or constraints. It is anticipated that these routing models will enable adaptive routing schemes to be used towards development of a truly flexible many-core usage model, whereby tasks can be added dynamically into the many-core system and the self-optimisation will allow the system to maintain homoeostasis through online adaptation. The power of the emergent behaviour has been demonstrated by showing that a

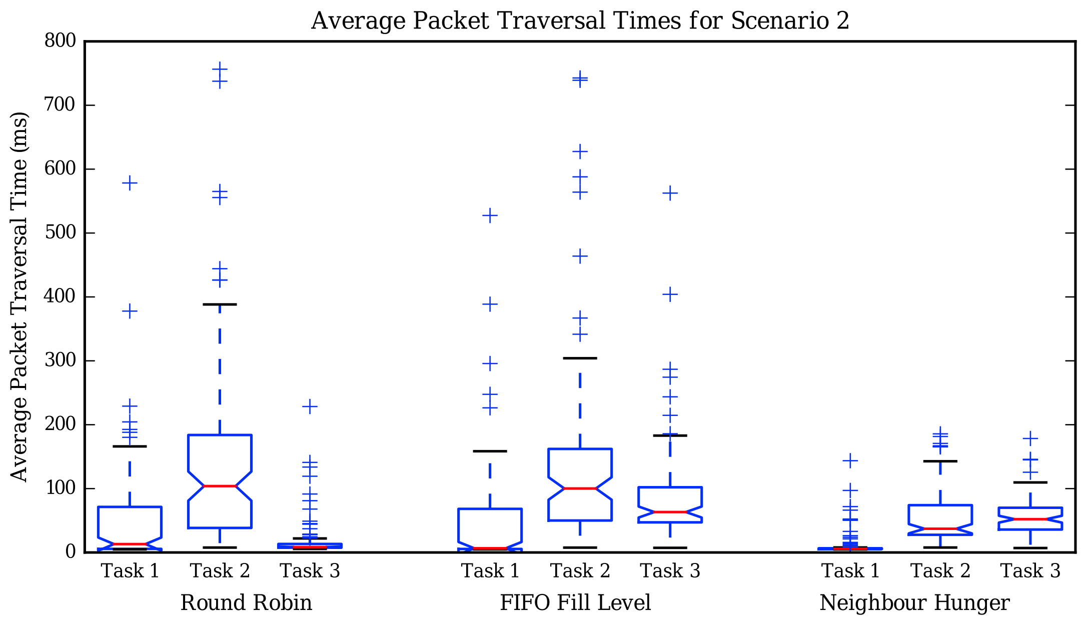

Fig. 6. The average traversal times for the second scenario shows a slightly different pattern due to the imbalance between tasks. As shown in Figure 4, task 1 has a relatively large number of task 2 nodes to send its packets to, whilst task 2 suffers from the inverse when the parallel tasks are all bought to task 3 nodes. The Hunger scheme however ensures that the load is better shared amongst all of the task 3 nodes and so exhibits a substantial improvement over the other schemes when routing packets from task 2 nodes. It is interesting to note that the FIFO Fill level scheme shows a greater range for packets sent from task 3 nodes, this is due to the large number of packets from task 2 in the network and the preference of the scheme to service the bottleneck buffers flooded with task 2 nodes, in turn bringing the range of task 2 packet latencies down slightly at the expense of task 3 packets.

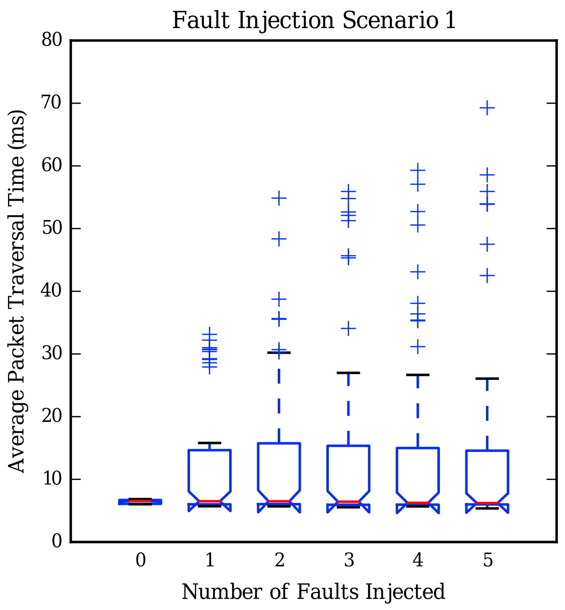

Fig. 7. Results of a fault injection experiment with the first application scenario. A constant topology had up to five random faults introduced before the experiment is then run for 5000 packets. The is repeated 25 times and the average packet traversal time for all tasks of each run is plotted.

fault tolerance scheme can be achieved by simple exploitation of the monitors that are attached to the system. This suggests that the sense-think-act model is a successful approach/tool for the implementation of scalable yet adaptively fault tolerant many-core systems.

VI. Discussion and Further Work

Autonomous adaptive "intelligent" routing was the main focus of this investigation, however, the results suggest, and the metaphor adopted ensures, that by simply including additional sensory inputs it would be possible to enhance the behaviour of each agent. The local "sense-think-act" model for each agent has been demonstrated to be very powerful when viewed as an emergent behaviour at the system level. Therefore, future work shall look at how this model can be extended by applicable behaviours from the social insects. Indeed the polytheistic nature of the social insects is an obvious starting point for experimenting with dynamic task allocation and swapping. The outliers in the experiments have shown how important the NoC topology is to effective routing and therefore is the next extension to the many-core self optimisation. With the right sensory inputs (for example ring oscillators, FIFO fill monitors), there is significant scope for extending the same intelligent task allocation in a way to also offer fault prediction as well as fault tolerance. This would allow a many-core system to autonomously optimise core task allocation whilst taking the temperature or degradation state of the device into

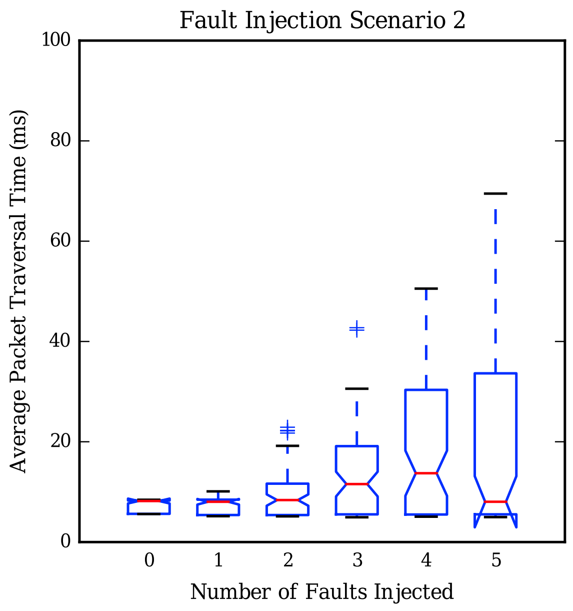

Fig. 8. Fault injection experiment results for the second application scenario. Due to the unbalanced nature of the second application model the location of a fault has a greater impact on the network performance (e.g. the case of a faulty task three node), and thus may account for some of the variation in scaling of the medians as the selection of faulty nodes is random.

account at runtime, as opposed to the complex analysis and simulation that is currently required to solve these issues.

When these experiments are considered in terms of the information exploited, we see a range between no information used in routing decision (Round Robin), local node knowledge used in routing decision (FIFO fill level) and neighbour knowledge used in decision (Hunger). By exploring the information exchange mechanisms used within social insect colonies we may find some other interesting paradigms for information sharing and exploitation across the array of nodes. For example these experiments could be expanded by adding a notion of packet priority to the system and the routers are responsible for deciding whether to change a packet's priority, for example a router could decide to increase the priority of a packet as it ages; this would reduce both the severity of outliers and the spread of the distribution and if combined with a deadline could be a technique for providing social insect inspired soft-realtime aspects to many-core systems. This would be an example of a decision at a single node (decision to increase the priority) affecting the behaviour of other, unknown nodes that process this packet at some later point in the many-core system.

References

[1] L. Benini and G. D. Micheli, "Networks on chips: a new SoC paradigm," *Computer*, 2002.

[2] A. Hemani, A. Jantsch, S. Kumar, and A. Postula, "Network on chip: An architecture for billion transistor era," *Proceeding of the IEEE NorChip Conference. Vol. 31.*, 2000.

[3] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 23–es, Aug. 2007.

[4] K. Srinivasan, K. Chatha, and G. Konjevod, "Linear programming based techniques for synthesis of network-on-chip architectures," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* IEEE, 2004, pp. 422–429.

[5] U. Ogras and R. Marculescu, "Energy- and Performance-Driven NoC Communication Architecture Synthesis Using a Decomposition Approach," in *Design, Automation and Test in Europe*. IEEE, 2005, pp. 352–357.

[6] G. Tempesti, "Graceful Design," *International Innovation Issue 140*, pp. 76 – 78, 2014. [Online]. Available: http://www.internationalinnovation.com/graceful-design/

[7] E. S. Shin, V. J. Mooney, and G. F. Riley, "Round-robin arbiter design and generation," in *Proceedings of the 15th international symposium on System Synthesis - ISSS '02*. New York, New York, USA: ACM Press, Oct. 2002, p. 243.

[8] M. Samie, G. Dragffy, and T. Pipe, "UNITRONICS: A novel bio-inspired fault tolerant cellular system," in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, Jun. 2011, pp. 58–65.

[9] M. Dorigo and L. Gambardella, "Ant colonies for the travelling salesman problem," *BioSystems*, 1997.

[10] T. Stützle and M. Dorigo, "ACO algorithms for the traveling salesman problem," *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*, 1999.

[11] L. Chittka and J. Niven, "Are bigger brains better?" *Current biology : CB*, vol. 19, no. 21, pp. R995–R1008, Nov. 2009.

[12] S. Beshers and J. Fewell, "Models of division of labor in social insects," *Annual review of entomology*, 2001.

[13] J. H. Sudd, *An introduction to the behavior of ants*. St. Martin's Press, 1967.

[14] A. Greensted and A. Tyrrell, "RISA: A Hardware Platform for Evolutionary Design," in *2007 IEEE Workshop on Evolvable and Adaptive Hardware (WEAH2007)*. IEEE, 2007, pp. 1–7.