A Methodology for Cognitive NoC Design

Wo-Tak Wu and Ahmed Louri

Abstract—The number of cores in a multicore chip design has been increasing in the past two decades. The rate of increase will continue for the foreseeable future. With a large number of cores, the on-chip communication has become a very important design consideration. The increasing number of cores will push the communication complexity level to a point where managing such highly complex systems requires much more than what designers can anticipate for. We propose a new design methodology for implementing a cognitive network-on-chip that has the ability to recognize changes in the environment and to learn new ways to adapt to the changes. This learning capability provides a way for the network to manage itself. Individual network nodes work autonomously to achieve global system goals, e.g., low network latency, higher reliability, power efficiency, adaptability, etc. We use fault-tolerant routing as a case study. Simulation results show that the cognitive design has the potential to outperform the conventional design for large applications. With the great inherent flexibility to adopt different algorithms, the cognitive design can be applied to many applications.

Index Terms—NoC, network-on-chip, cognitive process, intelligent agent, adaptive, machine learning, multicore, fault-tolerant

1 Introduction

The proliferation of multiple cores on the same die heralded the advent of a communication-centric system wherein the design of the on-chip communications connecting various modules has become extremely important. A flexible and scalable packet-switched architecture called network-on-chip (NoC) [5] is commonly used. It is expected that core count will continue to increase. With continuing advances in process technology, the core count is expected to soon exceed one thousand [3].

As the number of components of a system increases, the probability of having component failure also increases. Also, network congestions, deadlocks and hotspots may occur. There are many potential issues in operating the NoCs. They will reach an operating complexity level that cannot be anticipated by their designers. The network must be intelligent enough to cope with the unforeseeable conditions so that the entire system can function efficiently and reliably [6]. We propose to build cognitive NoCs (CogNoCs), i.e., networks that can recognize and adapt to changes in the environment, learn from experiences and retain knowledge gained for future usage. With such capabilities, the system can attain better latency, throughput, power consumption and fault tolerance.

Learning typically is a relatively slow process whereas router's operations, like the route computation (to find out where to forward the packet to), takes only one or two clock cycles. To accommodate such big discrepancy, we propose a hardware and software co-design approach to build CogNoC. In CogNoC, the routers and the cores are augmented with hardware (for the routers) and software (for the cores) that facilitate the slow learning process and the high-speed router operations. At the high level, the router operation in a CogNoC works as follows. The router recognizes the inputs as a pattern. It uses such pattern to look up a knowledge base (KB) in the router for a solution (a plan of actions). If a solution is found, it acts on it immediately; otherwise, it relies on the core to execute an algorithm to produce a solution, which will be updated to the KB. The core may also initiate a process to find a better solution in parallel with the router operations. Next time the same pattern occurs, the router will be able to find the solution in the KB immediately.

The router does not contain any hard-wired complex decision-making algorithms, for route computation or flow control, for example. This provides tremendous flexibility to allow the system to adopt different algorithms, thus widening the router's design space. Furthermore, CogNoC only relies on local information; there is no centralized control in the network. Each core-router pair works autonomously to achieve a global objective, i.e., efficient delivery of network packets. This has the potential for better network scalability.

In this work, we chose fault-tolerant routing as a case study. The network needs to route packets from source nodes to destination nodes in spite of faulty connections among routers. Simulation results show that compared to a conventional NoC, CogNoC is able to find shorter paths in 8 percent of all paths by using a simple learning algorithm and achieves an average improvement of 15 percent in latency.

2 Background

2.1 NoC

The prevailing topology is 2D mesh. Fig. 1a shows a 4×4 mesh. Each core is attached to a router which connects to four neighboring routers (except for those at the boundaries). Message from one core is routed through the routers to its destination. One commonly used routing algorithm is the dimension-order routing (DOR) [4]. When a packet reaches a router, the router determines which router it will relay the packet to. Fig. 1b shows the router micro-architecture. It has a set of buffers for holding the packet temporarily and a crossbar switch to direct the packet to a different direction (north, east, south or west). NoC's routing operation is typically broken into different stages and execute in a pipelined fashion at GHz clock rate [4].

2.2 Cognitive Process

Thomas [8] proposed the definition and framework of a cognitive network for an adaptive data network (not a NoC) that can observe, act, learn and optimize its performance. The network involves a cognitive process, which is depicted conceptually in a simpler form in Fig. 2. The active component is the intelligent agent. It observes what is happening in the environment. The observations become the inputs to the agent. The agent processes the inputs with the knowledge it has. Then it produces a set of actions to apply to the environment to try to achieve the predefined system goals. Learning takes place when the desired outcome is not attained yet.

3 CogNoC Design

In this section, we lay out a set of requirements that CogNoC needs to satisfy. Then we provide a detailed architecture that satisfies these requirements. This set of requirements will guide the design and implementation.

3.1 Requirements

3.1.1 Speed

There are very stringent timing requirements on NoCs. In the router pipeline, it typically decides how to route the incoming packet in the first cycle, and the packet go through the pipeline in a few more cycles. So, speed is a critical factor in router operations. CogNoCs must be able to maintain the normal high-speed operation speed and at the same time accommodates the cognitive capabilities.

• *The authors are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721.*
E-mail: wotakwu@email.arizona.edu, louri@ece.arizona.edu.

Manuscript received 4 Apr. 2015; revised 6 May 2015; accepted 11 June 2015. Date of publication 18 June 2015; date of current version 23 June 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2015.2447535

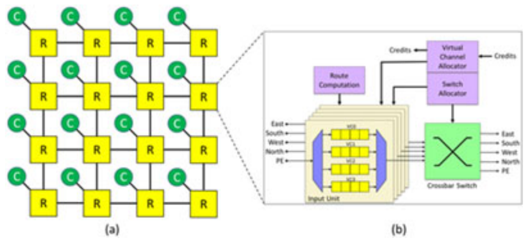

Fig. 1. (a). NoC of a mesh topology in a multicore architecture. Each core (C) is attached to a router (R). (b) Router microarchitecture.

3.1.2 Area

Silicon real estate is expensive in a multicore chip. It has significant impact on design, validation, manufacturing and testing. With additional functionality in a CogNoC, it is imperative to keep the router small. Thus the control logic implemented in a router cannot be too complex.

3.1.3 Autonomy

With a thousand of routers on-chip, a centralized control of all routers would not be feasible due to the large communication overhead. Router decisions must be made locally. We postulate that the property of autonomy will help produce effective solutions. This requirement is inspired by many elegant solutions observed in the natural world, e.g., in ant and bee colonies, where individual constituents act autonomously to achieve global goals without a centralized command structure. Most conventional NoCs are small and use very simple control schemes; autonomy is inherently achieved. For CogNoCs of much larger size and utilizing the cognitive capabilities, we need to maintain the same autonomous property.

3.1.4 Flexibility

Different applications may present vastly different network traffic patterns to the system. One single algorithm probably is not able to satisfy all different communication requirements. This requires routers to be flexible to run different algorithms. Conventional NoCs have only one single algorithm implemented as hard-wired logic in the router. However, the potentially many required algorithms in CogNoCs cannot be all realized the same way as they would take up too much area.

3.2 CogNoC Router Architecture

Routers in a CogNoC rely on cores for their operations. Therefore, part of the overall router architecture, depicted in Fig. 3, is the core. The router design is augmented with three modules: a pattern recognition module that translates the set of inputs (network packets and conditions) to a pattern that can be manipulated more efficiently; a KB module that stores the learned knowledge, which can be recalled in a single clock cycle; and a control agent orchestrates the router operations.

The pattern recognition greatly reduces the potentially vast input space to a much smaller pattern space. For example, an input is a 16-bit integer; we might want to reduce it to a qualitative entity

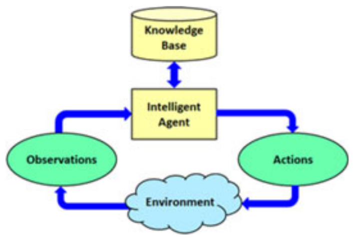

Fig. 2. The basic components in a cognitive process.

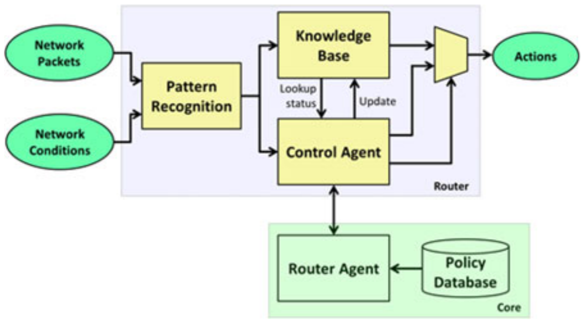

Fig. 3. CogNoC router architecture (only the control section of the router and router-related parts of the core are shown).

(a pattern), like *large*, *medium* and *small*, that can be encoded in just 2 bits. Such mapping is obviously application dependent. The algorithm might be quite sophisticated in that it can combine multiple inputs as one entity. This is a necessary step to limit the size of the KB, making the implementation practical.

The KB module must be very fast for high-speed router operations. For certain commonly known patterns, the KB can be preloaded to reduce the learning or computation efforts during run time. Basically, if the input pattern has been dealt with before, a known action plan can be retrieved immediately from the KB that the agent can execute on. Otherwise, the agent will need ask the core to produce a solution, which will be eventually updated to the KB. Next time that the same input pattern is encountered, the control agent knows how to handle it locally.

In the core, the router agent produces solutions for the router. A request from the router control agent wakes up the router agent in the core to compute an initial solution to be returned to the router immediately. The router can continue its operation. At the same time, the core router agent initiates the learning process to try to produce a better solution. If a better solution is found, it will be updated to the KB to replace the original solution.

A policy database in the core stores different algorithms and various rules or policies for generating new knowledge. The core router agent works closely with the operating system to determine what algorithms to use and what rules to apply, depending on the application. The determination can be made directly with specific inputs when a new application is started, or the core router agent can run a high-level monitoring process for making the decisions on an on-going basis.

3.3 Advantages of CogNoCs

With the KB built into the router, the router can maintain the normal high-speed operations except in the cases where the input patterns have not been observed before. Those cases will incur significant overhead in the communication with the core and the computation done by the software router agent. However, we pay the penalty only once. The introduction of the KB is inspired by the insight that the computation for the solution to a particular pattern needs take place only once; it is not necessary to compute again and again as a conventional NoC would do.

With the learning capability in a CogNoC, the solution may get better over time whereas it will stay the same in a conventional NoC.

The control logic in the router becomes simpler because the complex decision-making algorithms are contained in the core as software components, not as logic gates taking up space in the router. CogNoC does not waste time and energy to re-compute the same solution; it just looks up from the KB. The addition of KB incurs just a small area. As a result, we achieve overall reductions in router area and also power consumption (both static and dynamic).

The pattern recognition module and control agent do add some area overhead to the router. However, the pattern recognition

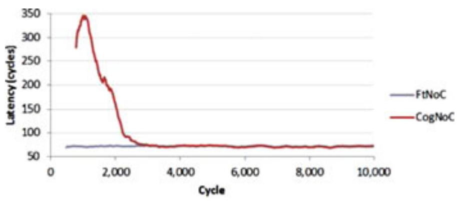

Fig. 4. Latency comparison over an extended period of time.

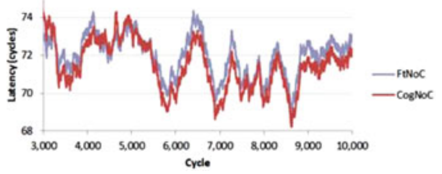

Fig. 5. CogNoC performs slightly better after initial learnings.

module can be made very simple and smaller, as illustrated in our case study, and the control agent's logic is rather simple. This extra overhead is easily offset by the overhead if the complex algorithms were implemented in the router.

Operation decisions are made only between the core and the router; there is no central control or management involved. All core-router pairs in the network work autonomously towards the global goal of efficient delivery of packets. The property of autonomy enables the NoC to be highly scalable because the router operations are not affected by the number of nodes in the network.

Since the majority of the logic is software-based in CogNoCs, it makes the system highly flexible in adopting various algorithms. Furthermore, the flexibility makes field upgradability and repair possible. It would be much easier to upgrade or repair products already deployed in the field because the software components can be replaced. A CogNoC design will also save costs in silicon validation and testing.

4 Implementation

We implemented the proposed CogNoC on gem5, a cycle-accurate multicore simulator [1]. We chose an application to illustrate the concept, namely fault-tolerant routing, which involves only the route computation stage in the router pipeline. The routers need to route the packets despite a number of faulty links in a large mesh network.

Gem5 only supports DOR, an algorithm assuming no faulty links in the network, so we added a fault-tolerant routing algorithm named F-cube4 [2]. However, F-cube4 does not have any learning capability, so we implemented reinforcement learning, a machine learning technique.

For very large and complex systems, it is difficult or infeasible to determine precisely at design time the model under which they operate. Attaining an optimal solution cannot be guaranteed given the vast search space and working constraints. We only use a best-effort approach to provide a solution which may be sub-optimal.

4.1 Router

The pattern recognition module is a simple preprocessing module to extract the general direction where the packet should be heading from the packet's destination node's coordinates. It is a logical choice to map the coordinates to four quadrants. The routing algorithm is oblivious to the exact destination of the packet; it only considers its general direction expressed as quadrant. The health state of a link can be represented by one bit. Combining the quadrant, link health states and routing state of the packet, an input pattern is thus formed for further processing. The pattern can easily be fitted into a 12-bit data item.

The conventional routing table in a router is replaced with the KB. It is implemented as content-addressable memory (CAM) [4]. Given the pattern produced by the pattern recognition module, the KB searches for a match in the memory to produce a solution or a set of actions. If a match is found, the solution consists of 1) the direction where the packet need be forwarded to; 2) the output virtual channel number; 3) updated routing state. The total solution width is less than 8 bits long. As the input width increases, the probability of having invalid combinations of fields within the input pattern also increases. A KB implementation with regular memory would require full memory capacity, thus a lot of the capacity will be unused. CAM provides the flexibility to support only valid patterns, and at the same time limit the memory capacity.

The KB is preloaded with solutions to the normal no-faulty-link conditions to avoid the route computation. The KB issues an output signal to indicate whether the input pattern is matched or not. If matched, the KB outputs a set of actions; otherwise, the control agent will send a request to the core for a solution.

4.2 Core

The router agent in the core is a software component. It executes the F-cube4 algorithm to produce an initial solution, an alternative direction to route the packet, which is updated to the KB and used to forward the packet. Then the packet continues its journey.

Due to the path diversity of the mesh network, directions other than the initially chosen one may work better. The core router agent initiates a learning process to explore alternative directions. There is a cost function associated with the path of delivering a packet to the destination. The function being used is the hop count (number of routers encountered in the path). It will be used to select the best solution to keep, i.e., the one with the lowest cost.

To find the cost of a path, the agent sends out a special *ping* control packet to the destination node. As the *ping* packet traverses through the network, it tallies the routers it encounters. Upon reaching the destination node, the count is returned back to the sender core.

There may be multiple faulty links encountered in the path. Since each core-router pair operates independently, all the learning processes initiated take place in parallel in different core-router pairs. Thus, the CogNoC can converge to a final solution rather quickly.

5 Evaluation

We run simulations with synthetic random traffic loads on a 16×16 mesh network. The injection rate per node is 0.01 packet/cycle. Each node sends 1,000 packets to randomly selected nodes. Faulty links are modeled on links between two routers. A faulty link means packets cannot be transferred between the routers directly. Twenty percent of all router-router links are designated as faulty before simulation starts. The core router agent is not cycle-accurately simulated at the moment. We estimate that the algorithm execution and the communication overhead total to about 100 router clock cycles. We compare the latency performance of CogNoC with that of a non-cognitive design, i.e., conventional design with just addition of the F-cube4 algorithm, which we will label as FtNoC hereon.

The simulation results show that CogNoC is able to shorten 8 percent of all routes with an average reduction of 15 percent in hop counts on those paths, compared to FtNoC. Fig. 4 shows that initially, the average latency of CogNoC is much higher than FtNoC because of learning new paths. After about 3,000 cycles (3 μs for 1-GHz clock rate), the latency of CogNoC becomes almost the same as FtNoC, as almost all the necessary learnings have been completed. Fig. 5 highlights the same figure at the low latency range, showing CogNoC performs a bit better, by an average of 0.7 cycles, than FtNoC after the learning period.

6 Related Work

Liotta [6], Thomas et al. [8] and Martinez and Ipek [7] have proposed to build data networks or multicore systems with cognitive approach or machine learning techniques. Our proposed work targets specifically on large-scale NoC design. To the best of our knowledge, this is the first time that cognitive concepts have been used in NoCs using a hardware and software co-design approach.

7 Conclusion

We have devised a methodology for designing and implementing cognitive NoCs. CogNoCs have ability to learn and to recognize changes in the environment. Network constituents work autonomously but collectively achieve global goals. It is a holistic approach that crosses many architectural layers.

A case study on fault-tolerant routing shows that with a simple and straight-forward learning algorithm, CogNoC outperforms slightly the conventional design over an extended period of time. It is able to quickly overcome the initial learning's large overhead.

The combination of cognitive capabilities and architectural design makes CogNoC a promising approach to address challenging issues effected in ever increasingly complex systems. We believe the CogNoC approach is a good way to target many large applications that allow time for the system to overcome the initial learning overhead. CogNoC provides many benefits: 1) better performance over time; 2) simpler and smaller router design; 3) great flexibility to accommodate different types of algorithms, which broadens the design space and increases the applicability to many application domains; 4) better network scalability; 5) field upgradability and repair; and 6) reduction in costs in silicon validation and testing.

References

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011.

[2] R. Boppana and S. Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," *IEEE Trans. Comput.*, vol. 44, no. 7, pp. 848–864, Jul. 1995.

[3] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. 44th Annu. Design Autom. Conf.*, 2007, pp. 746–749.

[4] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[5] N. Jerger and L. Peh, *On-Chip Networks*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2009.

[6] A. Liotta, "The cognitive NET is coming," *IEEE Spectrum*, vol. 50, no. 8, pp. 26–31, Aug. 2013.

[7] J. Martinez and E. Ipek, "Dynamic multicore resource management: A machine learning approach," *IEEE Micro*, vol. 29, no. 5, pp. 8–17, Sep./Oct. 2009.

[8] R. Thomas, L. DaSilva, and A. MacKenzie, "Cognitive networks," in *Proc. 1st IEEE Int. Symp. Dynamic Spectrum Access Netw.*, 2005, pp. 352–360.