An Efficient Mapping Algorithm on 2-D Mesh Network-on-Chip with Reconfigurable Switches

Salih Bayar
Idea Teknoloji
R&D Center
İstanbul, Turkey
salih.bayar@ideateknoloji.com.tr

Arda Yurdakul
Boğaziçi University
Computer Engineering,
İstanbul, Turkey
yurdakul@boun.edu.tr

***Abstract*—As Network-on-Chips (NoC) are the most scalable architecture with growing number of processing elements in multi-core systems, communication data between nodes tend to travel through more routers. Hence, a good mapping algorithm must be designed in order to locate most communicating nodes neighbour to each other. However, this may not be sufficient for data-intensive applications such as audio, video, telecommunication and etc. In such multi-core applications, processing elements communicate to each other with heavy load statically in most cases. Passing heavy load data through routers might make the routers bottleneck of the system. In this paper, we propose a custom 2-D NoC architecture with simple reconfigurable switches, which can be configured during both design and runtime according to the application requirements. We designed a mapping algorithm which tries to set paths through these simple switches instead of complicated routers all the way. Experimental results show that our mapping algorithm reduces routing cost up to by 79.96% for real life embedded applications.**

***Index Terms*—FPGA, NoC, Mapping**

I. Introduction

Most of the applications on multi-core SoCs have non-uniform communication traffic patterns and they can be predicted statically [1]. In addition to this, most of the multi-core SoC applications do not have many different communication flows (number of edges in task graphs) and each of these cores mostly communicates with a few of other cores. Usually, the traffic flow of these applications is already known beforehand [2]. As a result, the network topology, mapping of cores onto the target architecture and also routing have significant impact on the overall system performance. Hence, we focus on mapping and routing algorithms on NoC architectures, which are mostly preferred in multi-core embedded SoCs.

Data-intensive applications which are mostly mapped on NoC architectures, have a small number of communication flows (related task graphs are mostly sparse) and suffer from area-inefficient, power-hungry routers. As the heavy load data must pass through routers, routers become the bottleneck of the system. There are various studies that try to suppress the drawbacks of routers used in both packet and circuit switched networks. Some of them are PNoC [3], DyNoC [4], ReNoC [5], Skip-links [6], Reconfig-Net [7] and RecoNoC [8]. None of these studies exploits the particle filters for either mapping or routing process. In order to suppress the negative performance effect of routers, some simple switching mechanisms can be used within the NoCs for sparse applications.

The organization of the paper is as follows; in Section II, reconfigurable NoC architecture is given in detail. Section III presents our mapping and routing algorithm for reconfigurable NoC architecture. In Section IV, we present our case studies and experimental results for various applications. Finally, Section V presents future directions and concludes the work.

II. NoC Architecture with Simple Switches

The target architecture, which is a mesh based 2-D reconfigurable NoC architecture is given in Figure 1. This reconfigurable architecture is inspired from the work proposed in [1]. Compared to a conventional 2-D mesh NoC architecture, it has additional simple configuration switches. In Figure 1, while rectangles represent processing elements attached to routers, circles are simple configuration switches. Switches are much simpler than conventional routers. They have simple switching capabilities and can be configured at design or runtime according to the communication requirement of a given application. In Figure 1, routers are connected to each other through the configuration switches. For example, a connection between cores 6 and 7 is set only through simple switches in Figure 1. On the right side, three possible switch configurations in different directions are presented as well.

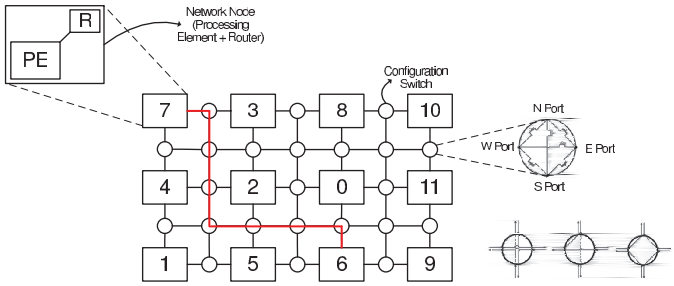


Fig. 1: 2-D reconfigurable NoC architecture with corridor width one (i.e. CW=1) [1].

The internal structure of a configuration switch is given in Figure 2. It has both inputs and outputs (N-bits, parametrizable) in each direction. Each switch is composed of 3-to-1 multiplexers at the outputs of each direction (North-

978-1-5090-0336-5/16/$31.00 ©2016 IEEE

East-South-West). Switching operations are achieved by these MUXs. Configuration information of these switches, i.e. select inputs of each MUX in switches are stored in separate Look-Up-Tables (LUTs) on the target device, i.e. Field Programmable Gate Array (FPGA). Additionally, there are 1-flit-size (e.g. 8-bits) buffers at each outputs that are used to increase clock frequency of communications for long links in a pipelined manner. More detail about these switches can be found in [1].

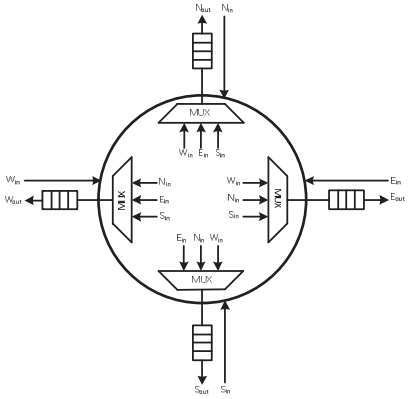


Fig. 2: Configuration switch internal structure.

In Figure 3, the reconfigurable 2-D mesh NoC architecture with the corridor width two (i.e. CW=2) is given. Increasing corridor width would be useful, if there is no solution for a given application for the given corridor width. With increased corridor width, traffic flows have more flexibility to traverse through the configuration switches.

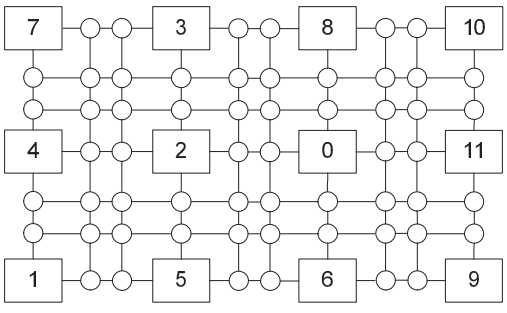


Fig. 3: 2-D reconfigurable NoC architecture with corridor width two (i.e. CW=2) [1].

III. Mapping and Routing Algorithm

The main objectives of our mapping and routing algorithms for the target NoC architecture are given as follows:

- Try to set connections through switches all the way.
- If sharing required, use minimum number of routers.
- Find an optimum mapping and routing by placing most communicating nodes close to each other.

The optimal solution for our algorithms can be defined as follows:

If the traffic flows between nodes are set only through the configuration switches and each node only communicates to its neighbours.

A. Main flow of mapping and routing algorithms

As described in our previous work [9], mapping process is one to one mapping of each separate task node of an application onto a physical core on the given architecture. Similarly, routing process can be defined as setting or creating physical paths through either configuration switches or routers between communicating cores.

Main steps of our mapping and routing algorithms are given in the order of occurrence as follows:

- Find an optimal mapping for a given task graph on the reconfigurable NoC architecture.
- Route communications according to the application requirements through switches and routers.
- Configure switches according to the routing information at design time or runtime.
- Generate NoC topology with reconfigurable switches.
- Switch-off/Remove unused switches and routers from the topology.
- Load sub-applications on the cores and start application on the target device.

As aforementioned, the routing process is applied to the final mapping found by our mapping algorithm.

B. Mapping algorithm

Definition 1: $Manhattan\ Distance\ (MDist)$ is the minimum number of hops from source node n_i to destination node n_j in the target NoC topology. $C_{r,c}$ is a physical core on the target NoC topology, who is located at row r and column c. The formula of the $MDist$ for the nodes $P_{r1,c1}$ and $P_{r2,c2}$ is given in Eq. 1.

$$MDist = |r1 - r2| + |c1 - c2| \tag{1}$$

The mapping cost, $MapCost$, of a configuration for a conventional 2-D mesh architectures is calculated for mapping only by using $MDist$ between each node pairs.

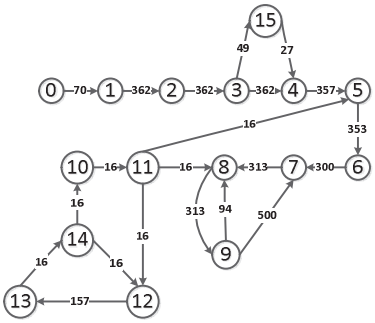
Definition 2: $t_{C_{r1,c1},C_{r2,c2}}$ is the traffic amount in 10Kbytes/s from source node n_i, which is mapped onto the physical core $C_{r1,c1}$, to destination node n_j, which is mapped onto the physical core $C_{r2,c2}$ in the target NoC topology.

The calculation of $MapCost$ for a single path (MC_{sp}) between processor nodes $C_{r1,c1}$ and $C_{r2,c2}$ in a configuration is given in Eq. 2.

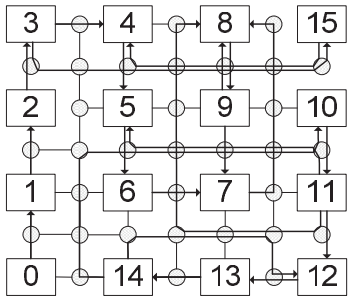
$$MC_{sp} = t_{C_{r1,c1},C_{r2,c2}} * MDist(r1, c1, r2, c2) \tag{2}$$

$MapCostTot$ (MC_{tot}) is the total communication cost of a configuration for a regular 2-D mesh topology as shown in Eq. 3. Here, R and C indicate number of total rows and columns in the target NoC topology respectively.

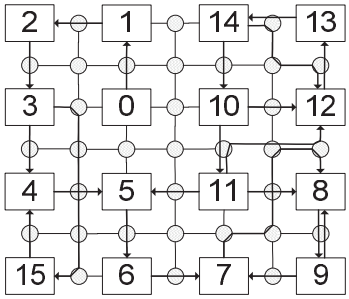
More detail about mapping algorithm can be found in [9].



(a) VOPD task graph with 16 nodes



(b) Routing with the work in AppAw[1] *(RoutCost=5753)*



(c) Routing after the PFMAP mapping[9] *(RoutCost=5243)*

Fig. 4: Routing of VOPD application with AppAw[1] and PFMAP [9] for CW=1

$$MC_{tot} = \sum_{r1=0}^{R} \sum_{c1=0}^{C} \sum_{r2=0}^{R} \sum_{c2=0}^{C} MC_{sp}(r1, c1, r2, c2) \quad (3)$$

C. Routing algorithm

Routing algorithm is very similar to the one given in [1].

Definition 3: Path is a dedicated link on the NoC topology, connecting source and destination nodes travelling through routers and switches.

$PathCost$ for the source node C_{r_s,c_s} and destination node C_{r_d,c_d} is given in Equation 4.

$$PathCost = \sum_{i=1}^{R+S} t_{P_{r_s,c_s},P_{r_d,c_d}} * RS_i \quad (4)$$

Here, RS_i is the cost of either a router or a switch on a path. R represents the number of routers and S represents the number of switch on a path. A simple configuration switch consumes about five times less power than a conventional router [1]. Total $RoutCost$ for a given configuration can be found in Equation 5. Here, E represents the number of edges in the given task graph.

$$RoutCost = \sum_{i=1}^{E} PathCost_i \quad (5)$$

The main objective here is setting paths for the heaviest communication flows at first. Mapping process has already given an effort to locate the most communicating nodes close to each other, i.e. with minimum hop counts. Hence, in routing process, we firstly sort communication flows in descending order. Besides, we check each edge in task graph, whether corresponding physical cores neighbour to each other. If the current communication flow, i.e. edge in the input task graph, is between neighbour cores, then we set these paths firstly. After setting all communication flows for neighbour cores in the descending order of communication volumes, we set paths for the remaining communication flows which are between non-neighbour cores.

IV. Case Studies

We developed and tested our mapping and routing algorithms in C++ with OPENMP library. We performed experiments with applications such as Video Object Plane Decoder (VOPD), MMS-Suite, Multi Window Display (MWD), Depth Map Computation (DMC) [10].

In Figure 4a, task graph of VOPD application and its mapping with routing on a 2-D mesh topology with both AppAw [1] (see Figure 4b) and PFMAP [9] (see Figure 4c) are given. The resulting mapping and routing of AppAw [1] illustrated in Figure 4b seems more complicated (most of switches and routers are used, i.e. there are overall connection paths between cores) than the result found by our mapping [9] and routing algorithm as shown in Figure 4c.

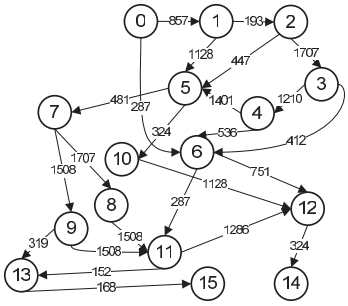
In Figure 5a, task graph of a synthetic application with 16 nodes is given. Figures 5b and 5c show final mapping and routing for AppAw and PFMAP respectively. Here, shaded rectangles show the routers in-use, black arrows show the direct connections through switches, blue arrows represent the unshared connections through routers and red arrows represent the shared connections through both switches and routers. Our routing after PFMAP mapping reduces *RoutCost* 44% and *the number of routers* 60%.

In Table I, $MapCost$ results are given. Since both MWD (with 12 cores) and VOPD (with 16 cores) are simple applications, improvements in these applications are not too much as for DMC (with 23 cores).

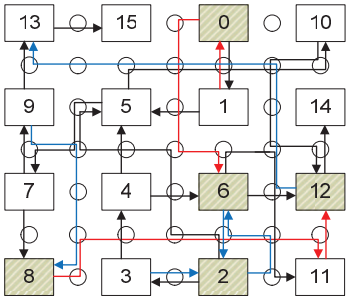
TABLE I: Mapping results for MWD, VOPD and DMC applications

Application	AppAw	PFMAP	PFMAP Imp. over AppAw
MWD	1248	1216	2,56%
VOPD	4265	4125	3,28%
DMC	14203	12393	12,74%

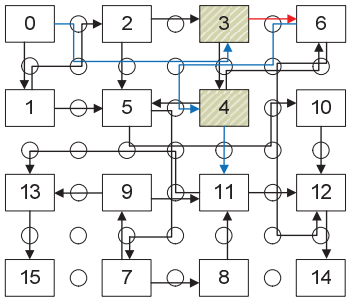
In Table II, $RoutCost$ results are given. PFMAP decreases routing cost up to 48,05% compared to routing found by AppAw. Here, it is clear that minimal improvements on mapping result in substantial improvements on routing.



(a) Synthetic task graph with 16 nodes from TGFF



(b) Routing with AppAw *(RoutCost=61092, RouterCount=5)*



(c) Routing after PFMAP mapping *(RoutCost=34258, RouterCount=2)*

Fig. 5: Mapping and routing of a synthetic graph with AppAw and PFMAP on the 2-D, 4x4, reconfigurable mesh NoC

TABLE II: Routing results for MWD, VOPD and DMC applications

Application	AppAw	PFMAP	PFMAP Imp. over AppAw
MWD	1632	1504	7,84%
VOPD	5753	5243	8,86%
DMC	114858	59666	48,05%

In Table III, $RoutCost$ results are given. MMS-Suite includes applications such as H263-Decoder, H263-Encoder, MP3-Decoder and MP3-Encoder. As these applications use the same set of IP-cores but the traffic pattern among the cores is different for each application, an average graph is used for both mapping and routing process. As explained in AppAw [1], the average graph is composed of all edges values of four input task graphs. PFMAP [9] reduces $RoutCost$ 30% to 35% for H263-Decoder, H263-Encoder and MP3-Encoder. However, for MP3-Decoder application, PFMAP [9] decreases routing cost up to 79.96%.

TABLE III: Routing results for MMS-Suite application

Application	AppAw	PFMAP	PFMAP Imp. over AppAw
H263-Dec	447721	311257	30,48%
H263-Enc	628608	409189	34,91%
MP3-Dec	212150	42520	79,96%
MP3-Enc	272893	181775	33,39%

V. Conclusion

In this paper, we developed task to core mapping and routing algorithms for a reconfigurable 2-D mesh NoC architecture. The objective of this work was using simple switches instead of heavy load routers for setting communication paths between task nodes on the given architecture.

We tested our mapping and routing algorithms on various data-intensive applications such as VOPD, MMS-Suite, MWD, DMC [10] and some of synthetic task graphs. Here we showed that our algorithms give better results than AppAw [1] in all these tests.

As a future work, the scalability of our mapping and routing algorithms can be investigated in detail. In order to do this, fully synthetic task graphs (generated by Task Graphs for Free (TGFF) [11]) with various NoC sizes for 2-D and 3-D mesh/torus can be used.

Acknowledgment

This work is fully supported by Idea Teknoloji R&D Center and State Planning Organization of Turkey, (DPT) under the TAM Project, Grant No. 2007K120610

References

[1] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Application-aware topology reconfiguration for on-chip networks," *IEEE Transactions on VLSI Systems*, vol. 19, no. 11, pp. 2010–2022, 2011.

[2] ——, "Virtual point-to-point connections for nocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 855–868, 2010.

[3] C. Hilton and B. Nelson, "Pnoc: A flexible circuit-switched noc for fpga-based systems," *Computers and Digital Techniques, IEE Proceedings -*, vol. 153, no. 3, pp. 181 – 188, 2006.

[4] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "Dynoc: A dynamic infrastructure for communication in dynamically reconfugurable devices," in *Int. Conf. on Field Programmable Logic and Applications*, 2005, pp. 153 – 158.

[5] M. Stensgaard and J. Sparso, "Renoc: A network-on-chip architecture with reconfigurable topology," in *2nd ACM/IEEE Int. Sym. on Networks-on-Chip*, 2008, pp. 55 –64.

[6] S. Hollis and C. Jackson, "Skip the analysis: Self-optimising networks-on-chip (invited paper)," in *Int. Symp. on Electronic System Design*, 2010, pp. 14 –19.

[7] J. Ma, C. Wang, Y. Wen, T. Chen, W. Hu, and J. Chen, "Dynamic reconfigurable networks in noc for i/o supported parallel applications," *Int. Conf. on Comp. and Inf. Technology*, vol. 0, pp. 2768–2775, 2010.

[8] R. Vancayseele, B. Farisi, W. Heirman, K. Bruneel, and D. Stroobandt, "Reconoc: A reconfigurable network-on-chip," in *6th Int. Workshop on Reconfigurable Communication-centric Systems-on-Chip*, 2011, pp. 1 – 2.

[9] S. Bayar and A. Yurdakul, "Pfmap: Exploitation of particle filters for network-on-chip mapping," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 2116–2127, Oct 2015.

[10] T. Schonwald, A. Viehl, O. Bringmann, and W. Rosenstiel, "Distance-constrained force-directed process mapping for mpsoc architectures," in *5th Euromicro Conference on Digital Systems Design*, 2012, pp. 592–599.

[11] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," http://ziyang.eecs.umich.edu/ dickrp/tgff.