Evaluating the feasibility of network coding for NoCs

Leandro Soares Indrusiak
Real-Time Systems Group, Department of Computer Science
University of York
York, United Kingdom

Abstract—**Network coding is a novel technique that can increase network throughput by linearly combining data packets in intermediate nodes and recovering them before delivery at their destinations. This paper discusses the applicability of network coding to NoCs and evaluates the potential advantages of that technique when supporting multicast communication.**

I. INTRODUCTION

Networks on chip (NoCs) are now a well established research topic and have several implementations available commercially. This level of maturity was achieved by a significant amount of original research work, together with techniques that were carefully adapted from decades of research and development in general-purpose networks. This paper attempts to adapt and evaluate one of such techniques, namely network coding, which has been successfully used in wired and wireless networks with different objectives such as increased throughput, robustness, or low power consumption. The specific focus of the paper is to use one particular strategy to perform network coding, based on butterfly networks, aiming to improve throughput in NoCs with 2D-mesh topology.

Butterfly networks exist only when multicast communication patterns arise, so the paper starts by reviewing typical multicast scenarios and related support by existing NoC architectures. Then, the foundations of network coding are explained and adapted to the particulars of a multi-hop mesh-based NoC architecture. Original contributions are presented on forming and evaluating butterfly arrangements within NoCs, as well as the experiments that were undertaken to corroborate the theoretical contributions.

II. BACKGROUND AND RELATED WORK

A. Multicast Communication in NoCs

In a multicast communication pattern, the network must efficiently transfer data from a single source to multiple destinations. This scenario appears frequently in content distribution networks, where one server sends the same set of data packets (e.g. video streams, stock quotes) to multiple clients, so several mechanisms were proposed to achieve efficient usage of the network resources [6].

There are several scenarios where we can observe single-source-multiple-destination communication in NoCs. Typical examples include cache coherency protocols in chip multiprocessors with shared memory, and embedded MPSoCs where sensor readings are processed by multiple software tasks running on different processors (such as speed, GPS location and proximity sensors in autonomous vehicles).

Just as it happens with routing, flow control and arbitration, most multicast mechanisms designed for regular networks are not applicable to NoCs because of its constraints in area and power consumption. A number of approaches that consider the particular constraints of NoCs are already available.

In [7], Samman et al proposed a simple multicast mechanism based on packets with multiple headers, one for each of its destinations. The authors also presented a router architecture implementing the proposed mechanism on a NoC with a mesh-of-fat-trees topology. The simplicity of the proposed multicast protocol required a modest increase of the router area (in comparison with the one without multicast support), but it also limited the gains on network throughput because of synchronous packet replication: the payload of all packets of a multicast will stall every time the header of any one of them is blocked.

Lu et al presented in [6] a mechanism based on the definition of destination groups and on the use of virtual channels to reserve a path for the multicasting of data to all destinations within a group. This approach has predictable delivery times once the path to the destination group is established, but the setting-up time is not negligible, specially if new multicast groups have to be created often. Other issues include finding paths that reach all destinations while maintaining the network deadlock-free and the fact that network nodes at the end of the multicast path will probably experience much larger latencies than they would in the case of unicast.

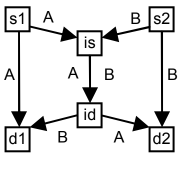
Recursive partitioning multicast was proposed in [8], and is based on a packet replication policy which is recursively applied at each node. It partitions the network in different regions, and checks whether there are destination nodes at each of those regions. If positive, it then decides which output ports should be used for each replica of the packet. Asynchronous packet replication is used, so the problems experienced by [6] are possibly avoided, but this was not discussed in detail by the authors in their experimental evaluation.

B. Network Coding

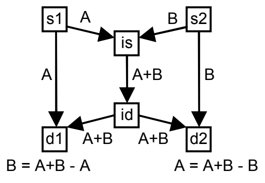
Network coding is a novel field in information theory that has been applied in different application domains, from underwater wireless sensor networks [4] to peer-to-peer file distribution [3]. It is based on the principle that a network node can combine a number of incoming packets into a (possibly different) number of outgoing packets, instead of simply forwarding them [1]. The technique that is used by a given node to combine packets into new ones is called coding, and different choices of coding can impact the network in different ways, resulting in increased network throughput or robustness, for example.

This paper investigates the application of linear network coding to on-chip networks, aiming to achieve an increase on network throughput and a reduction in power consumption. Linear network coding was proposed by Li et al [5] and states that the data sent on the outgoing channels of a given node is a linear combination of the data received on its incoming channels. Unlike packet concatenation, a linear combination of packets of size L will also have size L. Thus, encoded packets sent out by a given node carry information about several of the original incoming packets, but each individual outgoing packet does not provide means to allow the recovery of any of that information [1].

However, a destination node is able to decode a packet that was produced by linearly coding two packets A and B, in case it also receives either one of the original packets (for ease of understanding, think that the linear coding function could simply be a bitwise XOR of the packet's payload). This scenario was often exemplified in the context of butterfly networks, which is a simple case of multicast communication, and then generalized for different scenarios and application domains. Fig. 1, which is based on similar diagrams from [1] and [2], depicts two variations of such a scenario. In both variations, each of the two source nodes appearing at the upper right and left corners of the butterfly sends packets to both destination nodes at the lower corners. In the first variation, without network coding, it is not possible to perform both multicasts without time-multiplexing the link between the nodes in the centre of the butterfly. A similar conclusion can be reached for different routing protocols, possibly changing the link(s) that must be multiplexed. Using network coding, however, all receivers can be served simultaneously by transmitting a coded packet through the central link. Both destinations are able to use the coded packet to recover the data they need, since each of them has also received one of the original packets.



(a)



(b)

Figure 1. Butterfly network without (a) and with (b) network coding.

III. BUTTERFLY NETWORKS ON CHIP

This paper presents a first attempt to evaluate the applicability of network coding to NoCs. Therefore, it is not supposed to address unlikely scenarios or unusual system architectures, but to focus on typical cases instead. Thus, the chosen starting point is a butterfly network based on linear network coding, as covered in Section II.B, mapped to a NoC with 2D-mesh topology.

Network coding can increase the throughput of a butterfly network by substituting two packets by a single coded one that can be decoded by two destinations. In a butterfly network, sources are directly connected to one of the destinations and one of the intermediate nodes. Likewise, destinations are also directly connected to the remaining intermediate node. While such arrangement may appear within a 2D mesh, it is not the only possibility to construct a butterfly over that topology. Considering the multi-hop nature of NoCs, each of the arrows in Fig. 1 can represent any number of hops in a 2D mesh. For example, Fig. 2 shows different butterfly networks over an 8x8 mesh.

Such extended definition of a butterfly topology differs from the original definition of [1] in two critical points:

- for a given set of two sources and two destinations, it may be possible to find several different butterflies;

- for a particular set of two sources and two destinations, it may be impossible to find a butterfly which can increase the communication throughput using network coding.

The first point can be illustrated by the different butterflies connecting the sources s_1, s_2 with destinations d_1, d_2 in Fig. 2 (one of them through the intermediate nodes i_a, i_b and another through i_c, i_d).

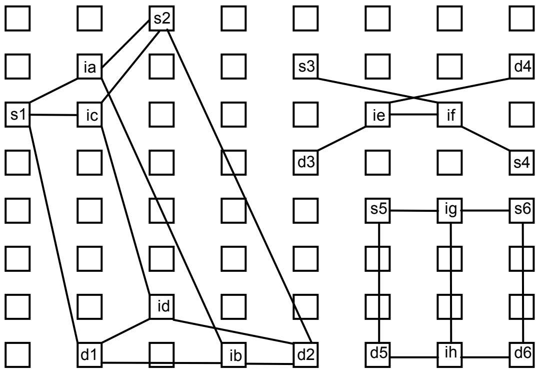


Figure 2. Butterfly arrangements over a 2D mesh. The lines connecting nodes are simply illustrative, and a real implementation of the connections would follow a particular routing algorithm over the structure of the mesh.

The second point will be addressed in more detail later, but the butterfly connecting sources s_3, s_4 with destinations d_3, d_4 in Fig. 2 already provides some insight into that situation, as the number of hops needed to construct that butterfly is larger than the number of hops needed to directly connect each source to each destination. Considering that all links have the same capacity, the route with a higher number of hops will have lower throughput.

Taking into account both points identified above, it is possible to divide in two parts the problem of evaluating the applicability of network coding to NoCs:

- how to find an optimal butterfly arrangement, given a set of sources and destinations;

- how to evaluate whether a given butterfly can potentially increase the communication throughput between sources and destinations.

A. Butterfly formation

First, the definition of a butterfly arrangement and its number of hops is given. A constraint is also given, in order to ensure that the butterfly retains the shorter segments between sources and destinations.

Definition 1: A butterfly B is formed by connecting six nodes s_1, s_2, d_1, d_2, i_s and i_d through the following segments, denoted by their endpoints: s_1d_1, s_2d_2, s_1i_s, s_2i_s, i_si_d, i_ds_1, i_ds_2.

Definition 2: The number of hops of a segment m_in_j denoted by $hops(m_in_j)$ is given by the Manhattan distance between m_i and n_j.

Definition 3: The number of hops of a butterfly $hops(B)$ is the sum of the hop count of its composing segments: $hops(s_1d_1)+hops(s_2d_2)+hops(s_1i_s)+hops(s_2i_s)+hops(i_si_d)+hops(i_ds_1)+hops(i_ds_2)$.

Constraint 1: A butterfly must fulfil the condition: $hops(s_1d_1)+hops(s_2d_2) \leq hops(s_1d_2)+hops(s_2d_1)$. If that is not the case, the labels of s_1 and s_2 must be swapped so that the constraint is met.

The problem of finding an optimal butterfly arrangement, given the sources and destinations, is then defined as the problem of finding its intermediate nodes. In this case, the optimal butterfly is the one with the lowest hop count, as that is the one with the highest potential throughput.

Problem 1: Given four nodes s_1, s_2, d_1, d_2, find two intermediate nodes i_s and i_d so that the butterfly formed by them has the lowest possible hop count.

An intuitive solution to this problem is to consider i_s as the midpoint between s_1 and s_2 and, likewise, i_d as the midpoint between d_1 and d_2. This solution is straightforward and produces acceptable results to well-formed butterflies, such as the one connecting s_5, s_6, d_5 and d_6 in Fig. 2. However, in some cases it is not possible to find a single midpoint because of the discrete nature of the coordinate space of a mesh (in Fig. 2 for instance, both nodes between d_1 and d_2 can be considered as midpoints). For those cases, one of them could be arbitrarily selected. In this paper, the one with lowest x and y coordinates is chosen, as shown in Fig. 3.

```
midpoint(p1,p2){    x = (p2.x + p1.x) /2;
                      y = (p2.y – p1.y) /2;
                    return new point(⌊x⌋, ⌊y⌋); }
```

Figure 3. Pseudocode for the *midpoint* function.

Fig. 5 shows an example of a butterfly whose intermediate nodes were calculated using the *midpoint* function defined in Fig. 3. By observing different situations, one can realize that in many cases a butterfly would have a lower hop count if the intermediate nodes would not lie within the line between sources and destinations, but somewhere on its adjacency. A constructive heuristic was then devised, aiming to find the intermediate nodes in those cases. It uses the routing algorithm of the particular NoC on which it is applied, denoted by the function *nextHop,* to iteratively find new candidates for the intermediate nodes, starting from the original sources and destinations. It also uses the midpoint function defined in Fig. 3 as a means to evaluate whether the candidate nodes are acceptable: if their hop count to the opposite midpoint is lower or equal to the hop count of the previous candidate, they are acceptable.

An improvement to this constructive heuristic was also devised, by updating the opposite midpoint at each iteration. The original heuristic calculated it once upon initialization, so it required only two calls to the midpoint function. The improved one will call it once per iteration until it converges, aiming to better results but at a higher computational cost. The pseudocode for the improved heuristics is shown in Fig. 4.

```
1   $i_{st}$ = midpoint($s_1$, $s_2$);  $i_{dt}$ = midpoint($d_1$, $d_2$);
2   $i_{s1}$ = $s_1$; $i_{s2}$ = $s_2$; $i_{d1}$ = $d_1$; $i_{d2}$ = $d_2$;
3   while($i_{s1} \neq i_{s2}$ OR $i_{d1} \neq i_{d2}$){
4      $p_1$ = nextHop($i_{s1}$, $i_{s2}$);
5      if( hops($i_{s1}i_{dt}$) $\geq$ hops($p_1i_{dt}$) ) { $i_{s1}$= $p_1$;}
6      $p_2$ = nextHop($i_{s2}$, $i_{s1}$);
7      if( hops($i_{s2}i_{dt}$) $\geq$ hops($p_2i_{dt}$) ) { $i_{s2}$= $p_2$;}
8      $p_3$ = nextHop($i_{d1}$, $i_{d2}$);
9      if( hops($i_{d1}i_{st}$) $\geq$ hops($p_3i_{st}$) ) { $i_{d1}$= $p_3$;}
10     $p_4$ = nextHop($i_{d2}$, $i_{d1}$);
11     if( hops($i_{d2}i_{st}$) $\geq$ hops($p_4i_{st}$) ) { $i_{d2}$= $p_4$;}
12     $i_{st}$ = midpoint($i_{s1}$, $i_{s2}$);  $i_{dt}$ = midpoint($i_{d1}$, $i_{d2}$);
13  }
14  $i_s$ = $i_{s1}$; $i_d$ = $i_{d1}$;
```

Figure 4. Pseudocode for the constructive heuristics.

As the improvement is restricted to the line 12, the first constructive heuristic can also be seen by disregarding that line. Fig. 5 also shows examples of butterflies whose intermediate nodes were calculated using the regular and the improved constructive heuristics, allowing for some comparison: the regular heuristic constructed a butterfly with hop count 29, while the one created by the improved heuristic had 28 and the basic midpoint approach produced one with 31. A proper comparative analysis of the three heuristics will be presented in Section IV.

B. *Butterfly evaluation*

The use of butterfly networks is by no means a guaranteed optimisation of multicast communications between two sources and two destinations, posing the following problem:

Problem 2: Given four nodes s_1, s_2, d_1, d_2, and two intermediate nodes i_s and i_d forming a butterfly network, compare the total hop count of the butterfly segments with the total hop count of the equivalent unicast communications between sources and destinations.

Once a butterfly is constructed, the solution of Problem 2 is trivial, because it is simply a comparison between the results of two series of sums. By solving Problem 2 for a few examples, one can already see that in some cases a butterfly arrangement will not always be able to increase the network throughput because in some cases the hop count of the butterfly segments will be higher than the regular sequence of unicasts from sources to destinations. For some cases, this will happen no matter how good the butterfly formation heuristics are, so it is important to single out the cases which can not be improved by constructing butterflies.

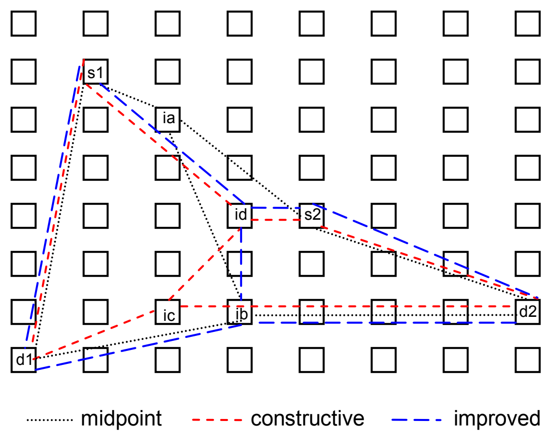


Figure 5. Results of different butterfly formation heuristics.

Given the computational cost of each of the heuristics, the ideal situation would be a test that could be quickly applied before the butterfly is formed. Referring back to the definition of Problem 1, this means that such test must use only the location of nodes s_1, s_2, d_1 and d_2 within the mesh.

Problem 3: Given four nodes s_1, s_2, d_1, d_2, test if it is impossible to construct a butterfly network that has lower hop count than its unicast counterpart.

Constraint 1 ensures that segments s_1d_1 and s_2d_2 are the ones that remain after the formation of the butterfly, which means that the new segments linking the intermediate nodes actually replace the s_1d_2 and s_2d_1 segments. Thus, the test should simply be a comparison between the hop count of the new segments with the sum of the hop counts of s_1d_2 and s_2d_1. As the new segments can only be determined after the application of butterfly formation heuristics, a slightly different approach is required.

Observing a large number of different cases in which the butterfly formation is not effective, one notices that the sources and destinations are often located at the edges of a cross - i.e. the paths from sources to destinations would cross at one point close to the epicentre of the polygon formed by the four nodes. The distinctive feature of that shape is that the hop count between sources and destinations is larger than the hop count of the segments that the butterfly arrangement would substitute, namely s_1d_2 and s_2d_1.

Furthermore, the minimum possible number of hops between two sources (or destinations) and their intermediate node is equal to the Manhattan distance between them, regardless of the actual location of the intermediate node. Thus, it is possible to state that a butterfly arrangement will never have a lower hop count than its unicast counterpart if Condition 1 below is false:

Condition 1: $hops(s_1s_2)+hops(d_1d_2)<hops(s_1d_2)+hops(s_2d_1)$.

As the attentive reader may have inferred from the definition of Problem 3 and the phrasing of the previous paragraph, Condition 1 is considered *necessary* but not *sufficient* when referring to a possible reduction of the hop count of a butterfly network when compared to its unicast counterpart. If it evaluates false for a given set of sources and destinations, it means that no butterfly would ever be found with a lower hop count than its unicast counterpart, thus solving Problem 3. However, if it evaluates true, it does not necessarily confirms the existence of such butterfly, or that a particular butterfly formation heuristic can find it.

IV. Experimental Results

An experimental setup was created to corroborate the findings reported in Section III. The experiments were based on NoC models with 2D mesh topologies of different dimensions (3x3, 6x6, 9x9, 12x12) and a deterministic routing algorithm (XY dimension-order).

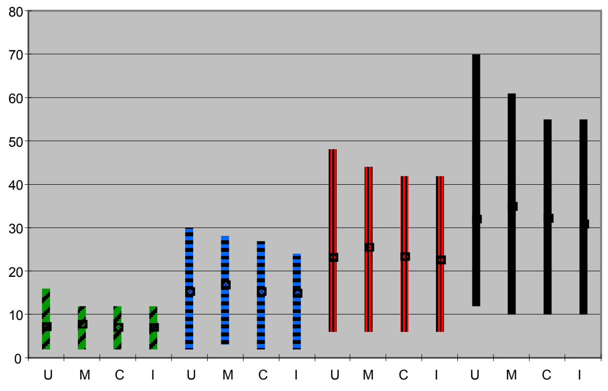
All three heuristics proposed in Section III.A were implemented and used to construct butterfly arrangements for a large number of randomly selected sets of sources and destinations. Fig. 6(a) shows, for different NoC dimensions, the maximum, minimum and average hop count of 1000 butterflies constructed by the different heuristics from different sets of sources and destinations. It also shows, for sake of comparison, the hop count needed by a sequence of unicast packages for the same sets of sources and destinations (each source directly sending one packet to each destination). The results are modest, showing a reduction of 10 to 25% on the maximum hop count, but only 2 to 5% on the average hop count.

However, those experiments disregarded Condition 1 and formed butterflies for each and every one of the 1000 randomly generated sets of sources and destinations. A similar experiment was also undertaken, but applying Condition 1 to decide whether a butterfly should be constructed. The result was that in 35% to 50% of the cases a butterfly was built, and that the higher percentage correlates to the higher dimensions of the NoC. Fig. 6(b) shows the results of this experiment, where the reduction on the maximum hop count was of 18 to 22% and on the average hop count was of 10 to 14%.

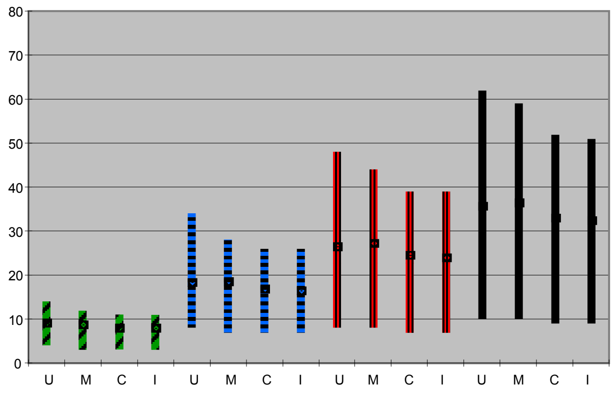
V. Conclusions

This paper presented a first approach to evaluate the applicability of network coding to NoCs. It considered coding within a typical butterfly arrangement, proposed heuristics to construct butterflies within 2D meshes, and identified a condition to determine whether a useful butterfly can be found for a given set of sources and destinations. Extensive experiments were performed, and the results fully corroborated the theoretical findings.

However, this work is only a first attempt to tackle this problem, so it does little more than touching its surface. A host of implementation issues were let aside and must be addressed by future work. Similarly, advanced network coding schemes were not considered and will certainly make the applicability of network coding to NoCs even more attractive. Another point worth exploring is the joint application of network coding and other coding techniques used to reduce power consumption in NoCs, such as [9].



(a)



(b)

Heuristics: U – unicast M – midpoint C – constructive I – improved

NoC topologies: 🟩 3x3 🟦 6x6 🟥 9x9 ■ 12x12

Figure 6. Experimental results – maximum, minimum and average hop count for 1000 multicast communications (two sources, two destinations) using different butterfly formation heuristics and over different NoC topologies. Vertical bars show the maximum (top of the bar), minimum (bottom of the bar) and average (black square) hop count obtained by each heuristic for the complete experiment. Baseline is the unicast communication between each source and each destination.

References

[1] R. Ahlswede, Ning Cai, S. Li, and R. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, 2000, pp. 1204-1216.

[2] C. Fragouli, J.L. Boudec, and J. Widmer, "Network coding: an instant primer," *SIGCOMM Comput. Commun. Rev.*, vol. 36, 2006, pp. 63-68.

[3] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, IEEE, 2005, pp. 2235-2245 vol. 4.

[4] Z. Guo, P. Xie, J. Cui, and B. Wang, "On applying network coding to underwater sensor networks," *Proc. 1st ACM International Workshop on Underwater Networks*, Los Angeles, CA, USA: ACM, 2006, pp. 109-112.

[5] S. Li, R. Yeung, and Ning Cai, "Linear network coding," *Information Theory, IEEE Transactions on*, vol. 49, 2003, pp. 371-381.

[6] Z. Lu, B. Yin, and A. Jantsch, "Connection-oriented Multicasting in Wormhole-switched Networks on Chip," *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, IEEE Computer Society, 2006, p. 205.

[7] F.A. Samman, T. Hollstein, and M. Glesner, "Multicast parallel pipeline router architecture for Network-on-Chip," *Design, Automation and Test in Europe Conference and Exhibition*, Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 1396-1401.

[8] L. Wang, Y. Jin, H. Kim, and E.J. Kim, "Recursive partitioning multicast: a bandwidth-efficient routing for Networks-on-Chip," *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, IEEE Computer Society, 2009, pp. 64-73.

[9] A. Garcia-Ortiz, L.S. Indrusiak, T. Murgan, and M. Glesner, "Low-power coding for Networks-on-Chip with virtual channels," *Journal of Low Power Electronics*, vol. 5, Apr. 2009, pp. 77-84.