# A Method for Efficient Mapping and Reliable Routing for NoC Architectures with Minimum Bandwidth and Area

Haidar M. Harmanani and Rana Farah
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

***Abstract*—Network-on-Chip (NoC) is an on-chip communication methodology that has been proposed as an alternative to bus-based communication in order to cope with the increased complexity in embedded designs. This paper presents a method for assigning tasks to nodes in a *2-D* mesh, and for determining the nodes positions on the mesh using simulated annealing. The method proposes a new efficient routing algorithm that minimizes blocking while increasing bandwidth throughput. The method is implemented and various benchmarks are attempted.**

## I. Introduction

The increase in design complexity has made the reuse of embedded cores imperative in order to reduce the productivity gap. Embedded cores are *predesigned* and *preverified* intellectual properties (IPs) that maybe *soft*, *firm*, or *hard*, and are typically heterogeneous with respect to functionality, size, and communication bandwidth. It has been argued by many researchers that there is a need to meet the high communication requirements of large systems-on-chip (SoCs) using a scalable communication methodology [3]. *Network-on-Chip* (NoC) designs are emerging technologies that consist of a number of interconnected heterogenous devices that communicate over a scalable interconnection network by bringing packet-based communication paradigms on-chip. NoC provides several advantages including modularity, higher performance, better structure, and compatibility with core designs and reuse [13]. NOCs are organized using various topologies such as *mesh/torus*, *fat-tree* and *ring*. A 2D-mesh topology consists of nodes that are arranged in a rectilinear grid where each node is bi-directionally connected to its *top*, *bottom*, *left*, and *right* neighbors [6]. Each node includes a multi-port switch that is associated with a core.

The design of an efficient NoC that satisfies application-dependent constraints is a complex task that includes synthesizing the network topology, modeling the traffic characteristics, and setting design parameters. One of the difficult problems in NoC is the topological mapping of resources such that to optimize a certain objective function. The mapping problem is an instance of the constrained quadratic assignment problem, an $\mathcal{NP}$-hard problem. Another concern in NoC implementation is selecting an efficient routing strategy while providing freedom from deadlocks and livelocks. Routing algorithms maybe *deterministic* or *adaptive*. Deterministic routing requires less resources since the routing path is chosen before the packets are sent. However, it does not utilize the network to its full potential especially when there is a heavy load. Various static routing techniques have been proposed such as *xy*, *toggle xy*, and *weighted toggle xy* routing. In *xy* routing, a packet is routed first in the *x* direction and then along the perpendicular *y* dimension. The *xy* routing strategy can be applied to regular two dimensional mesh topologies and produces a deadlock free NOC minimal paths but at the expense of available bandwidth. Furthermore, *xy* routing does not work with irregular meshes since some links may lead to a dead-end [4]. In order to alleviate the latency associated with blocking, *adaptive* routing techniques were proposed.

### A. Related Work

The NoC mapping and routing problems have been tackled by various researchers [14]. Hu et al. [8] proposed a branch and bound mapping algorithm that maps a set of IPs onto a NoC architecture while minimizing the total communication energy. The performance constraint was handled via bandwidth reservation. Lei et al. [11] used genetic algorithms to map an application on a mesh-based NoC architecture so as to minimize the execution time. Murali et al. [13] addressed the NoC mapping problem that supports traffic splitting under bandwidth constraint with the aim of minimizing communication delay. Ascia et al. [1] proposed a multi-objective mapping to mesh-based NoC architectures. The approach was reformulated using genetic algorithms [2]. Bolotin et al. [4] extended the *xy* routing with hard coded paths for deadlock free communication in an irregular mesh topology. Hu et al. [9] described a non-minimal deadlock free routing algorithm for an irregular NoC mesh topology with regions. The approach was extended in [10] to include deadlock-free deterministic routing. Stochastic routing for fault-tolerance in NoCs has been discussed in [7], [15].

### B. Problem Description

This paper presents an efficient method for *mapping* applications to mesh-based architectures and *routing path allocation* while minimizing *area* and *bandwidth*. Formally, given an

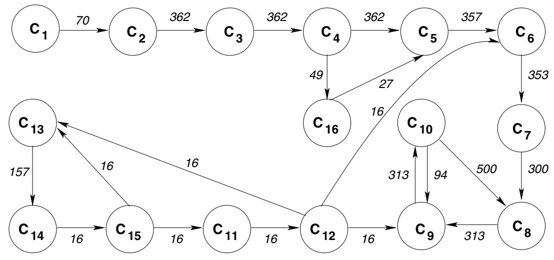

Fig. 1. Core communication graph for the video object plane decoder [13]

application with $N_c$ cores where each core has a specific area and bandwidth requirement, find a mapping of the cores to nodes in a 2-D mesh such that the total communication bandwidth is minimized subject to area constraint. The problem is tackled by 1) constructing a variable tile size mesh topology, 2) determining the cores position on the mesh such that the area constraint is respected, 3) routing path allocation such that blocking is minimized, and 4) minimizing the communication cost. The method is motivated by the following:

- NoC topological mapping is a combinatorial optimization problem;
- Routing path allocation is a difficult problem when guaranteeing freedom from deadlock and livelock. The proposed routing algorithm is livelock and deadlock free. It adapts to congestion and can sustain faults on links.

The remainder of the paper is organized as follows. Section II formulates the *mapping problem* using simulated annealing and presents our *routing algorithm*. The *annealing algorithm* is described in section III. We conclude with *experimental results* in section IV.

## II. Mapping and Routing Allocation in NoC

The mapping algorithm starts with a *core communication graph (CCG)*, $G = (V, E)$, which is a directed graph where each vertex $v_i \in V$ represents a core, and each directed edge $e_{i,j} \in E$ represents the communication between cores $v_i$ and $v_j$. The weight of edge $e_{i,j}$ represents the bandwidth and is treated as a flow of single commodities. Figure 1 shows the CCG for a video object plane decoder (VOPD) from [13]. In what follows, we formulate the mapping algorithm and describe the proposed routing method.

### A. Mapping Using Simulated Annealing

The key elements in implementing the annealing algorithm are: 1) the definition of the initial configuration, 2) the definition of a neighborhood on the configuration space, and a perturbation operator exploring it, 3) the cost function, and 4) a cooling schedule.

*1) Configuration Representation:* We propose an annealing configuration that is modeled using a two-dimensional vector where a *core* and a *switch* form a location (x,y) in the *2-D* mesh. Each core is connected to the communication network using a switch that is connected to adjacent switches. A cell can change its *core* and *switch* during the annealing iterations according to the neighborhood function.

```
Route ()
{
  Push the source router on the stack, $S_t$.
  while ($S_t$ is not empty)
    Pop a router, $R_i$
    If $R_i$ = Destination then
      route is found
  else if $R_i$ has been visited four times then
      mark $R_i$ as blocked.
  else {
    push $R_i$ on the $S_t$
    // check for next hop and make sure there are no loops in the route
    select a next hop $R_j$ based on priorities in Table I such that $R_j \notin S_t$
      if $\exists R_j$ such that it can accommodate traffic then
        push $R_j$ on $S_t$
      else
        mark $R_j$ as blocked.
  }
}
```

Fig. 2. Routing Algorithm

*2) Initial configuration:* The initial configuration is generated by randomly positioning the cores on the mesh.

*3) Neighborhood Function:* The algorithm explores possible mappings by randomly selecting two *tiles* and swapping their positions on the mesh. The process is repeated if the area constraint is violated.

*4) Cost Function:* Given $N_c$ cores, the objective is to find a mapping and a route path allocation that minimize the bandwidth. The communication cost of a communication channel is represented as a function of the length of the path and the potential blockage it may cause when it occupies a number of routing channels [5]. The cost function is to minimize:

$$\sum_{message} Number\ of\ Routes \times Bandwidth. \tag{1}$$

Subject to area constraint.

### B. Routing

We propose a routing algorithm that selects a route among alternative paths based on the network state and queues occupancy. Thus, if congestion is detected, packets will be routed in a congestion free path. The proposed algorithm uses a depth-first search on a tree with a maximal degree of 4, which is the number of adjacent hops a packet can move to. The algorithm, shown in Figure 2, is based on a maze methodology that has been proven to be effective in artificial intelligence applications where a *packet* is routed through a maze according to a set of rules and is *re-routed* only upon reaching a *congestion*, a *broken link*, or a *missing tile*. The algorithm implies a local knowledge of the network relative to the packet's position, and requires that upon encountering an intersection of routes in the network, one path is selected while "remembering" the intersection. If the selected path leads to a *dead-end*, due to congestion for example, then the packet is re-routed to the previous hop it encountered and follows another path until the final destination is reached. At each hop, a next hop is selected based on the position of the destination relative to the current position using the priorities shown in Table I.

| *Destination's position relative to source router* | *Next hop priority* | | | |
|---|---|---|---|---|
| | *First* | *Second* | *Third* | *Fourth* |
| Above | Up | Right | Left | Down |
| Above to the left | Up | Left | Right | Down |
| Above to the right | Up | Right | Left | Down |
| Below | Down | Right | Left | Up |
| Below to the right | Down | Right | Left | Up |
| Below to the left | Down | Left | Right | Up |
| Right | Right | Up | Down | Left |
| Left | Left | Up | Down | Right |

TABLE I
NEXT HOP SELECTION BASED ON CURRENT AND NEXT ROUTER POSITION

For example, if the destination is above the current router, then the packet is routed via a hop that is above the current router. However, the second time this packet is re-routed due to blocking, it will be routed to the right of the current router based on the first row in Table I. Thus, in the worst case, a router maybe visited four times before declaring the packet undeliverable. The list of priorities ensures that the packet is routed towards the final destination while guaranteeing that the route is deadlock free. The route is loop-less since each path is a depth-first search tree.

We illustrate our routing algorithm using the map for the VOPD example in Figure 3. Assume that a packet of size 100 bits is being sent from IP7 to IP9. Based on our algorithm, IP9 is below IP7 and to the right. Thus, the first hop is down to IP6. However, IP6 is already assigned a traffic of 780 bits and cannot accommodate the additional 100 bits. The algorithm backtracks, and the second choice based on Table I is to the right through $R_{13}$. $R_{13}$ is allocated 32 bits of traffic out of a maximum of 679 bits and can accommodate the additional 100 bits. $R_{13}$ is selected as the first hop; the traffic on $R_{13}$ is adjusted to 132 bits, and becomes the source router. For the second hop the priority is to go downward to $R_{12}$ which can accommodate the 100 bits, and $R_{12}$ becomes the source now. Finally, the next hop is downward to $R_9$ which happens to be the final destination. Please note that the above route is not indicated on the grid.

## III. Algorithm

The proposed algorithm, shown in Figure 4, starts by selecting an initial mapping and then performs a sequence of iterations. During each iteration, a new configuration is generated in the neighborhood of the original map by randomly selecting two *tiles* and swapping their locations on the mesh where an annealing configuration represents an intermediate grid with different bandwidth and area cost. The placement algorithm first determines the number of tiles in the mesh by selecting one cell and adding one column and one row such that the width of each cell in the column is equal to the width of the largest core on that column, and the height of the cell in a row is equal to the highest core on that row. The algorithm repeats the above process until all cores have been considered. The algorithm next places the cores on the map such that highly-communicating nodes are positioned close to each other, subject to area constraint. The routing

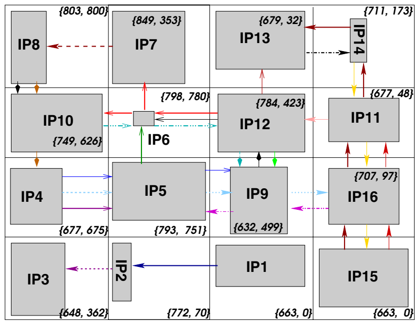

Fig. 3. Map and route path allocation for the video object plane decoder. Each cell is annotated with the router's *total* and *allocated* bandwidth.

```
Annealing_Mapping()
{
  MaxTime ← Total allowed time for the annealing process
  MaxArea ← Maximum allowed map area
  do {
      $M = M_0$
      do {
          do {
              NewMap = Explore_Neighborhood(CurrentMap)
              NewArea = Calculate_Area(NewMap)
              } while (NewArea > MaxArea)
          NewCost = Route(All_source_destination_pairs, NewMap)
          $\Delta_{Cost} = NewCost - CurrentCost$
          if ($\Delta_{Cost} < 0$) {
            CurrentMap=NewMap
            CurrentCost=Cost(CurrentMap);
            If ($NewCost < BestCost$) then {
              BestMap=NewMap
              BestCost = Cost(BestMap)
            }
          }
          else if random < $e^{-\frac{\Delta_{Cost}}{T}}$ then {        // $T$ = temperature
            CurrentMap = NewMap
            CurrentCost = Cost(CurrentMap);
          }
          $M = M - 1$
      } while ($M \geq 1$)          // $M$ is time until next parameter update
      $Time = Time + M_0$;
      $T = \alpha * T$;                       // $\alpha$ = Cooling rate
      $M_0 = \beta * M_0$;                    // $\beta$ = Iteration multiplier
  } while (Time < MaxTime);
  Return(BestMap);
}
```

Fig. 4. Annealing Mapping algorithm

path allocation is next performed and the variation in the cost function, $\Delta_{Cost}$, is computed. If $\Delta_{Cost}$ is negative then the transition from $C_i$ to $C_{i+1}$ is accepted. If the cost function increases, the transition is accepted with a probability based upon the Boltzmann distribution. The temperature is gradually decreased throughout the algorithm from a high starting value where almost every proposed transition, *positive* or *negative*, is accepted to a freezing temperature, where no further changes occur. The cooling schedule was experimentally determined as follows: $T_{init} = 40$, $\alpha = 0.99$, $\beta = 100$, and $M_0 = 5$. The algorithm repeats the above process for the maximum number of iterations, which was set to $1,000$.

## IV. Experimental Results

We implemented the proposed method using Java, and developed a multithreaded application where each core and

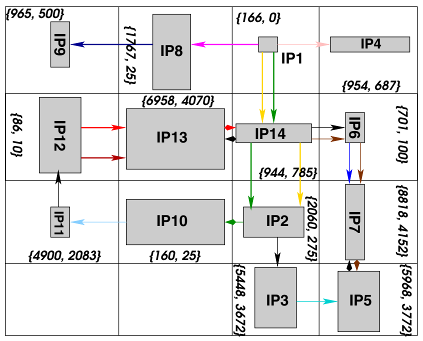

Fig. 5. Map and route path allocation for the *linearP13* example. Each cell is annotated with the router's *total* and *allocated* bandwidth.

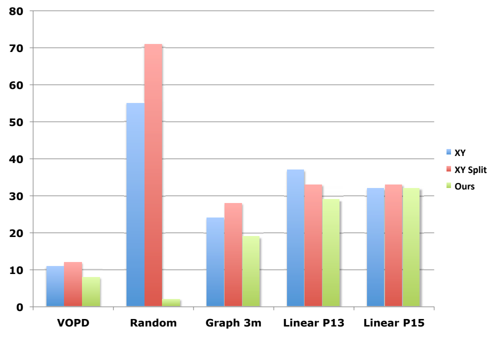

Fig. 6. Routing comparison results

a switch are implemented as an independent thread. Threads were interconnected in a *2-D* mesh based on our mapping and placement algorithm. We next used various benchmarks that vary in connectivity, communication weight, size, and occupation. The benchmarks include *graph2*, and *graph3m* designs from [12]; H.263 video encoder/decoder and mp3 audio decoder multimedia benchmarks from [16], that is *linearP13*, *linearP14*, and *linearP15*, and the *VOPD* and the *DSP Filter* examples from [13]. Figures 3 and 5 show the mapping results for *VOPD* and *linearP13* examples, respectively. The maps also illustrate the system level floorplan that was obtained by our system. Each node is annotated with the maximum bandwidth allowed by the router and the allocated bandwidth by our algorithm. Figure 6 clearly shows that our routing scheme outperforms the deterministic routing schemes in all attempted examples with respect to blocking and bandwidth. Table II shows the area and the bandwidth found by our algorithm for all attempted benchmarks. Furthermore, we have generated a random *core communication graph* with 60 nodes and generated random routes with random bandwidth such that 100% of the cores participate in the exchange. The results, shown in Figure 7, demonstrate that in the worst case, 9% of the communication channels were blocked, which is remarkable when compared to the number of blocked packets by the *xy* or the toggle xy routing algorithms. It should be also noted that the average number of hopes was comparable to the other routing algorithms when there is no blocking and at times was even better.

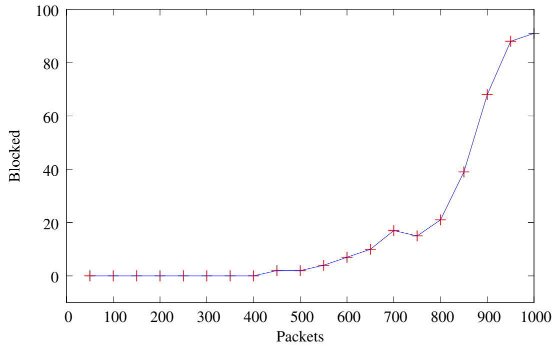

Fig. 7. Blocking in a random NoC where $N_c = 60$

## ACKNOWLEDGMENT

This work was supported in part by a grant from the Lebanese National Council for Scientific Research (CNRS).

## REFERENCES

[1] G. Ascia, V. Catania, and M. Palesi. Multi-Objective Mapping for Mesh-Based NoC Architectures. In *Proc. of the ICHSC/ICSS*, 2004.
[2] G. Ascia, V. Catania, and M. Palesi. A Multi-Objective Genetic Approach to Mapping Problem on Network-on-Chip. *JUCS*, 22(4), 2006.
[3] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli. Network-On-Chip Design and Synthesis Outlook. *Integration-The VLSI journal*, 41(2), 2008.
[4] E. Bolotin, A. Morgenshtein, I. Cidon, and A. Kolodny. Automatic and Hardware-Efficient SoC Integration by QoS NOC. *Proc. ICECS*, 2004.
[5] S. Bourduas, H. Chan, and Z. Zilic. Blocking-Aware Task Assignment for Wormwhole Routed Network-on-Chip. In *Proc. NEWCAS*, 2007.
[6] W.J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. DAC*, 2001.
[7] T. Dumitras and R. Marculescu. On-Chip Stochastic Communication. In *Proc. DATE*, 2003.
[8] J. Hu and R. Marculescu. Energy Aware Mapping for Tile-Based NoC Architectures Under Performance Constraints. In *Proc. ASPDAC*, 2003.
[9] J. Hu and R. Marculescu. DyAD-Smart Routing for Networks-on-Chips. In *Proc. DAC*, 2004.
[10] J. Hu and R. Marculescu. Energy and Performance-Aware Mapping for Regular NoC Architectures. *IEEE Trans. CAD*, 24(4), 2005.
[11] T. Lei and S. Kumar. A Two-Step Genetic Algorithm for Mapping Task Graphs to a NoC Architecture. *Proc. Euromicron Symp. on DSD*, 2003.
[12] C. Marcon, E. Moreno, N. Calazans, and F. Moraes. Evaluation of Algorithms for Low Energy Mapping onto NoCs. In *Proc. ISCAS*, 2007.
[13] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores Onto NoC architectures. In *Proc. DATE*, 2004.
[14] Umit Ogras, Jingcao Hu, and Radu Marculescu. Key Reserach Problems in NoC Design: A Holistic Perspective. In *Proc. ICHSC/ICSS*, 2005.
[15] M. Pirretti. Fault Tolerant Algorithms for Network-On-Chip Interconnect. In *Proc. Symp. on VLSI*, 2004.
[16] Krishnan Srinivasan, Karam S. Chatha, and Goran Konjevod. Linear-programming-Based Techniques for Synthesis of Network-on-Chip Architectures. *IEEE Trans. VLSI*, 14(4), 2006.

| *Benchmarks* | *Area* | *Bandwidth* | *# Cores* |
|---|---|---|---|
| graph2 | 90 | 4,061 | 10 |
| graph3m | 117 | 192 | 8 |
| linearP13 | 114 | 20,156 | 14 |
| LinearP14 | 156 | 255,440 | 12 |
| LinearP15 | 126 | 17,531 | 13 |
| Bertozi | 105 | 7,274 | 12 |
| VOPD | 270 | 5,705 | 16 |
| DSP Filter | 90 | 2,600 | 6 |

TABLE II
AREA AND BANDWIDTH RESULTS FOR ATTEMPTED BENCHMARKS