

# Application-Specific Buffer Space Allocation for Networks-on-Chip Router Design \*

Jingcao Hu  
Carnegie Mellon University  
Pittsburgh, PA 15213-3890, USA  
jingcao@ece.cmu.edu

Radu Marculescu  
Carnegie Mellon University  
Pittsburgh, PA 15213-3890, USA  
radum@ece.cmu.edu

## ABSTRACT

In this paper, we present a novel system-level buffer planning algorithm that can be used to customize the router design in Networks-on-Chip (NoCs). More precisely, given the traffic characteristics of the target application and the buffering space budget, our algorithm automatically assigns the buffer depth for each input channel, in different routers across the chip, to match the communication pattern, such that the overall performance is maximized. This is in deep contrast with the uniform assignment of buffering resources (currently used in NoC design) which can significantly degrade the overall system performance. For instance, for a complex audio/video application, about 85% savings in buffering resources can be achieved by smart buffer allocation using our algorithm without any reduction in performance.

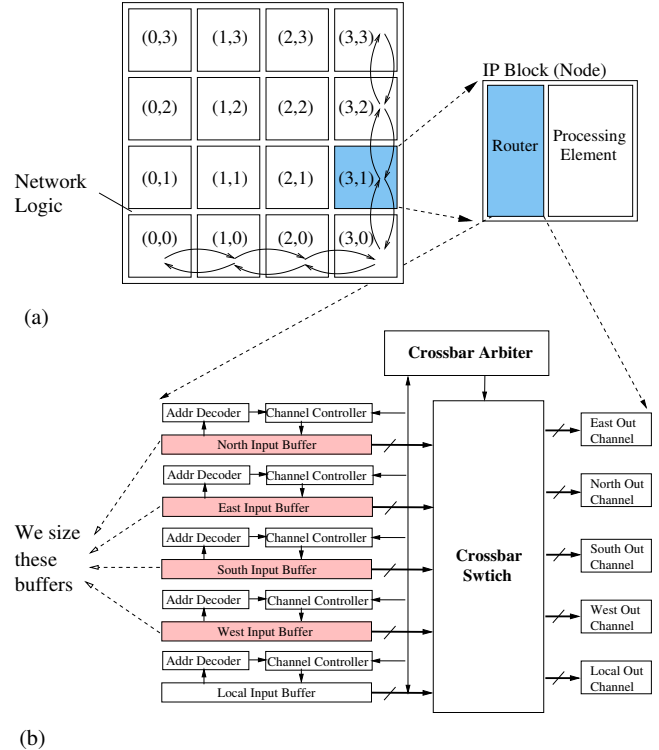
## 1. INTRODUCTION

With the advances in the semiconductor technology, it becomes possible for designers to integrate tens of Intellectual Property (IP) blocks together with large amounts of embedded memory on a single chip. This richness of the computational resources (CPU or DSP cores, video processors, *etc.*) places tremendous demands on the communication resources as well. Additionally, the shrinking of feature size in the *deep-submicron* (DSM) technologies makes interconnect delay and power consumption the dominant factors in the optimization of modern systems. Interconnect optimization under DSM effects is complicated because of the worsening effects due to crosstalk, electro magnetic interference, *etc.* [21].

The NoC approach was proposed as a promising solution to these complex on-chip communication problems [5][9][1][20]. For such an architecture, the chip is divided into a set of interconnected blocks (or nodes) where each node can be a general-purpose processor, a DSP, a memory subsystem, *etc.* Fig. 1(a) shows an example of a NoC implementation where nodes are connected in a 2D mesh topology. A router is embedded within each node with the objective of connecting it to its neighboring nodes (a typical on-chip router for a 2D mesh NoC is shown in Fig. 1(b)). As such, instead of routing design-specific global wires, the inter-nodes communication can be achieved by routing packets.

Compared to a standard data macro-network, an on-chip network is by far more resource limited. To minimize the implementation cost, the on-chip network should be implemented with very little area overhead. This is especially important for those architectures composed of nodes designed at a fine-level of granularity. The input buffers in a typical on-chip router (highlighted in Fig. 1(b)) take a significant portion of the silicon area of the NoC and consequently

\*Research supported by NSF CCR-00-93104 and Marco Gigascale Systems Research Center (GSRC).



**Figure 1: (a) NoC implementing a 2D mesh topology (b) Typical on-chip router architecture**

their size should be carefully minimized. On the other hand, the raw performance of a NoC is drastically impacted by the amount of buffering resources it can use, especially when the network becomes congested. Moreover, since the traffic characteristics varies significantly across different applications, the buffering resources have to be judiciously allocated to each input channel in order to match the specific communication patterns that characterize various applications. The uniform distribution of buffering resources, although straightforward and widely used in current NoC designs, fails to achieve this objective; this leads to poor performance and/or excessive use of the silicon area. To address such issues, the contributions of this paper are twofold:

- First, we propose an efficient algorithm which optimizes the allocation of buffering resources across different router channels, while matching the communication characteristics of the target application. More precisely, given the total available buffering space, the arrival rates between different com-

municating IP pairs and other relevant architectural parameters (e.g., routing algorithm, arbitration delay, *etc.*), our algorithm automatically decides the buffer depth for every input channel in each on-chip router. As an example, referring to Fig. 1, instead of uniformly assigning the buffer size to be four units in each and every input channel, we may assign the size of the *east input buffer* in the router located at tile (1,2) to be six while keeping the size of the *south input buffer* in the router located at tile (2,2) to be one. This can be done according to the communication characteristics of the target application, such that the overall network performance is maximized. For a complex audio/video application, such an application-specific buffer customization allows us to achieve the same performance level as a straightforward implementation which assigns the buffer size uniformly across the chip, but using 85% less buffering resources. This is also important from a power dissipation perspective.

- Second, we propose a novel analytical model which can be used to quickly analyze a given buffer size configuration and detect potential performance bottlenecks in the router channels. This is done by solving a set of nonlinear equations derived from detailed queuing models. This analytical model lies at the very heart of the algorithm for allocating buffer resources. The main advantage in using this analytical approach as opposed to straightforward simulation is the ability to quickly analyze the impact of various application-specific communication patterns on the overall system's performance.

The remaining part of this paper is organized as follows. In Section 2, we give a brief review of the relevant work. The problem of buffer allocation for NoCs is then described in Section 3. Following that, in Section 4, an efficient heuristic is proposed to solve this problem. Experimental results in Section 5 show that, compared to uniform buffer allocation, significant performance improvements can be achieved while satisfying the same total buffering space budget. Finally, we summarize our contribution and suggest possible directions for future work.

## 2. RELATED WORK

NoC design typically targets a specific application or a limited class of applications. Thus, the NoC architecture can be customized for each specific application in order to achieve best energy, performance and cost trade-offs [5][11]. There exists interesting work in this direction. In [2], Jalabert *et al.* show the benefits of the network topology customization on the system area, power and latency. The authors of [14][17] investigate the topological mapping of IPs onto the NoC architectures for bandwidth and communication energy savings. The work presented in this paper follows a different direction by addressing the customized, application-specific, allocation of buffer resources to different channels in each router. To the best of our knowledge, our work is the first to address the buffer allocation problem for NoC-based architectures and provide an efficient way to solve it.

Traditional work on performance evaluation in parallel computing either uses either time-consuming simulation (e.g., [12][6]) or provides analytical models for limited traffic conditions (typically uniform models) [4][7]. The assumption of uniform traffic makes sense in general purpose parallel computing as the interconnect architecture needs to support a wide spectrum of applications. However, such an assumption may not be appropriate for NoC designs; NoCs are typically designed for specific applications and thus exhibit very specific traffic patterns.

Adve and Vernon in [3] model a wormhole-based [8] mesh network as a closed queuing network and use approximate mean value analysis to calculate the average packet latency under arbitrary source-destination probabilities. However, all of these analytical models apply only to networks with infinite buffers (e.g., [4][7]) and/or single-flit buffers [3]. As such, they can not be used for buffer allocation in which the effect of arbitrary, but finite, buffer size has to be explicitly modeled.

## 3. THE PROBLEM OF ROUTER BUFFER ALLOCATION FOR NoCs

Simply stated, given the communication probability between each communicating IP pair and the total budget of buffering resources that the designer is allowed to use, the problem we need to solve is to find the *buffer depth assignment for each input channel*, across all the on-chip routers, such that the communication performance is maximized. If the performance is measured in terms of average packet latency, then maximizing the performance means, in fact, minimizing the end-to-end packet latency.

Next, we review the network platform and the traffic models which are relevant to our algorithm. We then formalize the problem and illustrate the significance of finding a good solution to it.

### 3.1 System characterization

#### 3.1.1 Platform characterization

The system under consideration is composed of  $m \times n$  tiles interconnected by a  $2D$  mesh network. Because of its simplicity, the choice of a  $2D$  mesh as the underlying NoC architecture serves only as an example for our algorithm<sup>1</sup>. We further assume that deterministic routing (e.g., dimension-ordered routing [18]) is used to direct the packets across the network instead of adaptive routing because of the resource limitations, as well as the out-of-order packet delivery problem associated with the adaptive routing. More precisely, store-and-forward or virtual cut-through [16] routing is assumed to be used for the network. Thus, in our analysis, a packet can be treated as a basic/atomic unit since it will always be transmitted or buffered as an indivisible entity.

For the sake of simplicity, we assume that all the packets in the network have a fixed size. Thus, in the absence of packet contention, the service time of each packet in a router (measured as the time span from the moment when the packet arrives at the header of the input channel of the router to the time it takes to receive it by the input channel of the downstream router) is fixed and can be accurately calculated. More precisely, the *service time per packet* ( $S$ ) in a router *without contention* can be calculated as follows:

$$S = T_{AD} + T_{SEL} + T_{ARB} + T_{CB} + T_{LINK} \quad (1)$$

In Eq. (1),  $T_{AD}$ ,  $T_{SEL}$  and  $T_{ARB}$  are the delays of the address decoding, routing path selection and crossbar arbitration, respectively. These parameters are usually independent of the packet length. On the other hand,  $T_{CB}$  and  $T_{LINK}$  model the delays in the crossbar and link traversal, respectively; they are usually proportional to the packet size. We note that different router architecture may lead to different delay models (e.g., [19]) but this will not change the flow of our buffer allocation algorithm.

We assume that the on-chip routers have the structure shown in Fig. 1(b). Each input controller has a separate buffer (typically implemented using registers for performance reasons) which buffers

<sup>1</sup>The algorithm can be extended for arbitrary topologies as it will be explained later.

the input packets before delivering them to the output channels. Each such input buffer in the router can have a different depth. This can be easily implemented, for instance, at the instantiation phase through a parameterized design methodology. When a new packet is received, the address decoder processes the incoming packet and sends the destination address to the channel controller; this controller determines which output channel the packet should be delivered to. Once the router has made the decision on which direction the data should be routed to, the channel controller sends the connection request to the *Crossbar Arbiter* in order to set up a path to the corresponding output channel.

The actual use of the input buffer is regulated through a back-pressure mechanism. Under this scheme, a packet is held in the buffer until the downstream router has enough empty space available (in the corresponding input buffer) such that network will not drop any packet in transit. This is extremely important for NoC architectures where implementing advanced end-to-end protocols may not be possible. Once a packet arrives at an input buffer, it will be served on a *first-come-first-served* (FCFS) manner.

Without loss of generality, we assume that the buffer size is measured in multiples of packet size. More specifically, let size of a packet be  $SP$  bytes; then the size of any input buffer must be  $m \times SP$ , where  $m$  is a positive integer. We also assume the size of the *local input* buffer (the input channel which accepts packets from the router's local PE) to be infinite. This is a reasonable assumption since the PE can also use its local memory (which is usually much larger compared to router buffers) to store input packets. Due to this assumption, the size of local input buffer will not be considered anymore in our allocation process.

The *Crossbar Arbiter* maintains the status of the current crossbar connection and determines whether or not to grant connection permission to the channel controller. When there are multiple input channel controllers requests for the same available output channel, the *Crossbar Arbiter* also uses the FCFS policy to decide which input channel gets the access, such that the starvation at a particular channel can be avoided.

### 3.1.2 Traffic characterization

Similar to most previous work on interconnection network performance evaluation, the traffic pattern of a given application is modeled assuming that each *processing element* (PE) injects packets with a *Poisson* distribution. More specifically, let  $(x, y)$  be the location of the tile at the column  $x$  and row  $y$  of the NoC as shown in Fig. 1(a); this means that for the PE located at  $(x, y)$ , the packet injection is modeled with a *Poisson* input rate  $a_{x,y}$ . Moreover, we assume that any packet injected in the network is independent. For a packet generated by PE at  $(x, y)$ , the probability that the packet is delivered to the PE at  $(x', y')$  is represented by  $d_{x,y}^{x',y'}$ .

The average packet latency ( $L$ ) is used as the metric for NoC communication performance. Similar to previous work, we assume that the packet latency spans the instant when the packet is created, to the time when the packet is delivered to the destination node, including the queuing time spent at the source. We also assume that the packets are consumed immediately once they reach their destination nodes. Although this is a simplified model, we feel that such simplifications are necessary in order to provide an analytical solution to the buffer allocation problem.

## 3.2 Problem formulation

For convenience, we summarize the basic parameters in Table 1<sup>2</sup>. With these notations, the problem of router buffer allocation for

<sup>2</sup>Some of the parameters will be explained later in more detail.

**Table 1: Parameter notation**

| Param.               | Description  |                                 |
|----------------------|--|---------------------------------|
| $SP$                 | The size of a packet   | Platform Specific Parameters    |
| $B$                  | The total buffering space budget   |                                 |
| $L$                  | The average packet latency   |                                 |
| $S$                  | Packet service time in a router without contention   |                                 |
| $\mathcal{R}$        | Routing function   |                                 |
| $R_{x,y}$            | The router located at tile $(x, y)$  |                                 |
| $C_{x,y,dir}$        | The $dir$ direction input channel in router $R_{x,y}$  |                                 |
| $PE_{x,y}$           | The PE located at tile $(x, y)$  |                                 |
| $l_{x,y,dir}$        | The buffer size of channel $C_{x,y,dir}$   | Application Specific Parameters |
| $a_{x,y}$            | Packet injection rate of $PE_{x,y}$  |                                 |
| $d_{x,y}^{x',y'}$    | The probability of a packet generated by $PE_{x,y}$ to be delivered to $PE_{x',y'}$          |                                 |
| $\lambda_{x,y,dir}$  | The packet arrival rate at channel $C_{x,y,dir}$   |                                 |
| $p_{x,y,dir}^{dir'}$ | The probability of a packet at channel $C_{x,y,dir}$ to be delivered toward $dir'$ direction |                                 |
| $\mu_{x,y,dir}$      | The service rate for packet at channel $C_{x,y,dir}$   |                                 |
| $\rho_{x,y,dir}$     | The utilization factor of channel $C_{x,y,dir}$  |                                 |
| $b_{x,y,dir}$        | The probability of the buffer at $C_{x,y,dir}$ being full                                    |                                 |

performance maximization under total buffering space constraints can be formulated as follows:

**Given:**

Total available buffering space  $B$

Application communication characteristics  $a_{x,y}$  and  $d_{x,y}^{x',y'}$

Architecture specific packet servicing time  $S$  and routing function  $\mathcal{R}$

**Determine:**

Buffer size  $l_{x,y,dir}$  for each input channel

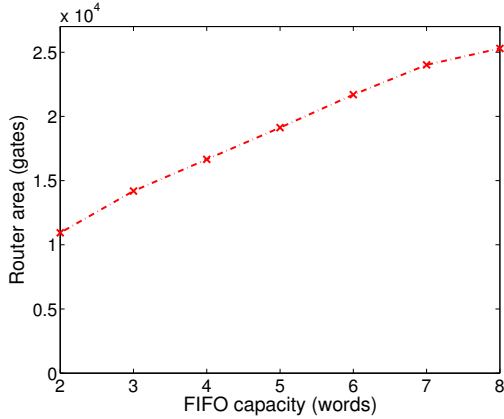
which **minimizes** the average packet latency  $L$ :

$$\min(L) \quad s.t. \quad \sum_{\forall x} \sum_{\forall y} \sum_{\forall dir} l_{x,y,dir} \leq B \quad (2)$$

## 3.3 Significance of the problem

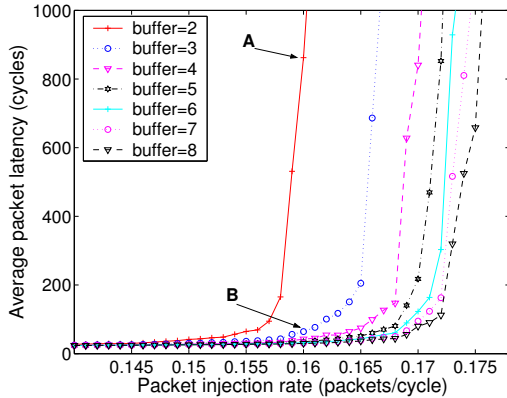
In order to be reduce the NoC implementation cost and power dissipation, small on-chip routers are definitely needed. To better understand the main resource consumers, we implemented a prototype of an on-chip router based on the architecture shown in Fig. 1(b) [15]. The router supports XY routing with FCFS crossbar arbiter and uses a 0.16 $\mu m$  technology. In this design, each input port has a fixed link width of 32 bits. The FIFOs are implemented using registers for good performance/power efficiency.

Fig. 2 shows the area of the router as a function of FIFO size. The X axis represents the FIFO capacity as number of words (a word is equal to 32 bits in this plot), while the Y axis gives the area of router in equivalent gates. As we can see, increasing the FIFO capacity significantly increases the router area. For instance, increasing the FIFO size in each input channel from 2 to 3 words leads to an increase of total router area by 30%. Thus, to minimize the implementation overhead of NoC, an effective approach is to reduce the overall use of the buffering space in the routers.



**Figure 2: Router sizes with different FIFO capacity**

To show the impact of the FIFO capacity on the communication performance, we simulated a  $4 \times 4$  NoC system under different traffic patterns when the FIFO capacity changes. Each simulation is first run for a warm-up period of 2000 cycles. After that, performance data is collected once 20,000 packets are sent. A cycle-accurate interconnection network simulator (*Nets*) was implemented in C++. *Nets* supports 2D mesh networks with packet routing; it has been designed to be easy customized to simulate different designs under various traffic patterns. Since many factors (e.g., routing path selection delay, crossbar arbitration delay, etc.) have a significant impact on the NoC performance, *Nets* models them accurately with the actual values taken from the prototype router design.



**Figure 3: System performance with different FIFO capacity**

Fig. 3 shows a typical latency/throughput plot under different communication loads and FIFO sizes. The simulated traffic pattern is *hot spot*<sup>3</sup>, where the PEs located at tiles (1, 0) and (2, 2) (see Fig. 1(a)) are the ones chosen to generate the two hot spots. The impact of the buffer size on the average packet latency is obvious from Fig. 3. More specifically, when the network becomes congested, increasing the buffer size helps in reducing the average packet latency. For instance, under the injection rate of 0.16 packets per clock cycle, the average packet latency of the system with a uniform buffer size of 2 is 862 clock cycles (point A in Fig. 3), while the average packet latency for a uniform buffer size of 3 is only 64 clock cycles (point B in Fig. 3).

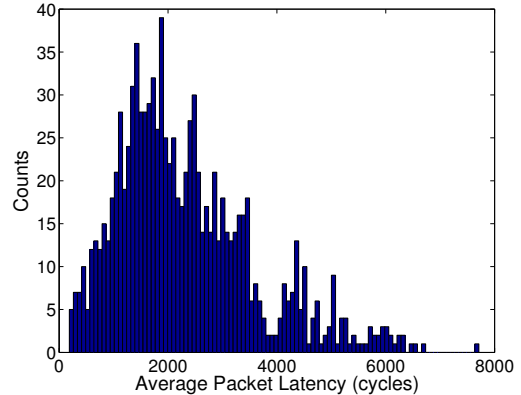
On the other hand, increasing *uniformly* the capacity of every FIFO in every on-chip router may not be the most effective way

<sup>3</sup>The *hot spot* traffic is explained in Section 5.

to use the silicon area. Because of the heterogeneity of the traffic pattern in most application-specific NoCs, it makes sense to allocate more buffering resources only to the heavy loaded channels. *The main idea is that the more “important” channels get allocated larger FIFOs compared to other, less important, channels.*

Indeed, it turns out that customizing the buffer size individually can improve the overall system performance significantly. To illustrate this idea, we arbitrarily select the system in which each input channel has a uniform buffer size of  $4 \times SP$  and use it as an example to see how much performance improvement can we gain by judicious buffer allocation. We note that the routers at the border of the NoC may have fewer input channels (for instance, the routers in the bottom row do not need south input channel). Thus, there are in fact only 48 channels and the total buffering space used is thus  $4 \times SP \times 48 = 192 \times SP$  (that is,  $B = 192$ ). We randomly generate 1000 solutions, all of them using  $192 \times SP$  buffering space. Each configuration is then simulated under the injection rate of 0.171 packets per second. The performance is shown as a histogram in Fig. 4.

As we can see, the solutions vary significantly in terms of average packet latency, despite the fact that all of them use the exact same total amount of buffering space. Another interesting thing to note is that, among the 1000 random generated solutions, the best solution ever found has average packet latency of 187 clock cycles, which is significantly better than the performance of the system having every buffer uniformly assigned of size  $4 \times SP$ , whose average packet latency is as high as 1856 clock cycles. (*There is a factor of 10 difference between the two latency values!*)



**Figure 4: Histogram for 1000 different random buffer configurations**

On the other hand, 1000 random solutions represent only a small portion of the entire solutions space, thus the optimal solution may perform even better than the best solution shown above. Indeed, by applying our buffer allocation algorithm, the system generated by the algorithm has an average packet latency of as low as 29 clock cycles at the injection rate of 0.171 packets per cycle. We need to note that the performance of the system with the customized FIFO buffers performs even better than the system with uniform FIFO capacity of  $8 \times SP$ , which has an average packet latency of 91 clock cycles at this injection rate. Consequently, by customizing the buffer size, we can actually implement a system which has better performance, but uses only half of the buffering resources compared to a system with FIFO size uniformly assigned to  $8 \times SP$ . This fully justifies the approach we take for FIFO size customization.

## 4. SOLVING THE ROUTER BUFFER ALLOCATION PROBLEM

In this section, we present a novel buffer allocation algorithm which starts from the minimum buffer size configuration (where each input channel has a buffer size of only one packet) and iteratively increases the buffer size of the bottleneck channels until the specified value of the buffer budget is reached.

### 4.1 Router/channel analytical models

The main part of the algorithm implements a technique for detecting the performance bottleneck among the different router channels. More specifically, given the current buffer size configuration, the algorithm tries to identify the channels where adding extra buffering space leads to the maximum improvement in performance. One way to guide this process would be to simulate the system implementing such a configuration and then profile the simulation results. Unfortunately, despite its flexibility, the simulation approach suffers from extremely long simulation times. Since we need to evaluate the system for every possible buffer configuration, the evaluation based on direct simulation is simply impossible to afford.

In the following, we propose a novel analytical model which can be used to quickly analyze the current buffer size configuration and detect the performance bottlenecks in the router channels; this is done by solving a series of nonlinear equations derived from queuing models. The basic idea is that, given the system configuration (which includes the traffic pattern, the routing delay and the size of each FIFO in the current solution), the algorithm detects the FIFO which has the highest probability to be in the “full state”. The channel which owns this particular FIFO becomes the real performance bottleneck in the current configuration and thus its size should be increased.

To solve this problem analytically, we resort to the theory of finite queuing networks [13]. The basic element in the model is a M/M/1/K finite queue (the first two “M” mean that the customer arrival time and server’s service time follow exponential distributions, “1” tells that the queue has one server to provide the service, and finally “K” represents the capacity of the queue). In this case, the channel  $C_{x,y,dir}$  is modeled as a finite queue of length  $l_{x,y,dir}$ , with the arrival rate  $\lambda_{x,y,dir}$ , served by one server with service rate  $\mu_{x,y,dir}$ . Both *inter-arrival* and *service times* are independent and identically distributed, following exponential distributions.

With this model, we can further develop the queuing model of a router as shown in Fig. 5. The five bubbles in the left hand side represent the five channels of  $R_{x,y}$  (that is, the router placed at tile  $(x,y)$ ), with  $N, E, W, S$  and  $L$  representing the directions of *north, east, west, south* and *local*, respectively. On the right hand side, the upper four bubbles represent the four corresponding input channels in router  $R_{x,y}$ ’s neighboring routers and the bottom bubble represents the output channel to  $R_{x,y}$ ’s local PE ( $PE_{x,y}$ ). These five bubbles on the right side give all the queues that the packets in  $R_{x,y}$  can possibly go to during the next time step and thus directly affect the calculation of the parameters related to  $R_{x,y}$ <sup>4</sup>.

Now let us consider, for instance, the north input channel at router  $R_{x,y}$ . This channel is represented as  $C_{x,y,N}$  in Fig. 5. Assuming the network is not overloaded (that is,  $\lambda_{x,y,dir} < \mu_{x,y,dir}$ ), then the arrival rate of  $C_{x,y,N}$  can be calculated using the following equation:

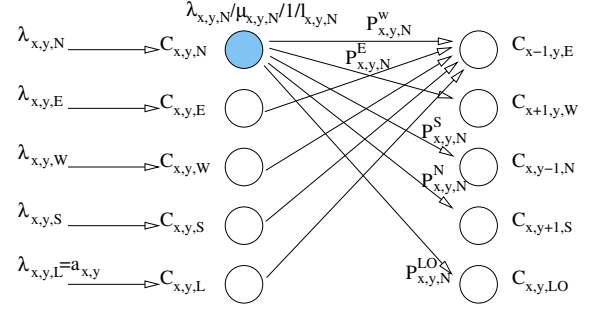


Figure 5: Queuing model of a router

$$\lambda_{x,y,N} = \sum_{\forall j,k} \sum_{\forall j',k'} a_{j,k} \times d_{j,k}^{j',k'} \times \mathcal{R}(j,k,j',k',x,y,N) \quad (3)$$

In Eq. (3), the routing function  $\mathcal{R}(j,k,j',k',x,y,N)$  equals 1 if the packet from  $PE_{j,k}$  to  $PE_{j',k'}$  uses the channel  $C_{x,y,N}$ ; it equals 0 otherwise. Note that we assume a deterministic routing algorithm, thus the function of  $\mathcal{R}(j,k,j',k',x,y,N)$  can be predetermined. Also, because the traffic flow is predetermined, the parameters  $p_{x,y,N}^W, p_{x,y,N}^E, p_{x,y,N}^S, p_{x,y,N}^N$  and  $p_{x,y,N}^{LO}$  can be pre-calculated. (We use  $LO$  to represent the *local output* direction, thus  $p_{x,y,N}^{LO}$  gives the probability of a packet in  $C_{x,y,N}$  to be delivered to  $PE_{x,y}$ .)

Now, the only unknown parameter for  $C_{x,y,N}$  is  $\mu_{x,y,N}$ . Once the value of  $\mu_{x,y,N}$  is determined, the probability of  $C_{x,y,N}$  to be in “full state” can be calculated straightforward using the finite M/M/1/K queuing model with the following equations [13]:

$$\rho_{x,y,N} = \frac{\lambda_{x,y,N}}{\mu_{x,y,N}} \quad (4)$$

$$b_{x,y,N} = \frac{1 - \rho_{x,y,N}}{1 - \rho_{x,y,N}^{l_{x,y,N}+1}} \times \rho_{x,y,N}^{l_{x,y,N}} \quad (5)$$

The calculation of  $\mu_{x,y,N}$  is not trivial, as it depends not only on the router’s service delay (as shown in Eq. (1)), but also on probabilities of a packet being routed to each downstream channel and whether or not the downstream channels are full. For instance, if the packet is to be delivered eastward and  $C_{x+1,y,W}$  is full, then the packet has to wait in  $C_{x,y,N}$ .

We derive the following models to take into consideration such effects. For the sake of simplicity, we assume next that the packet size is fixed and a link can transmit a packet within one clock cycle. This assumption can be relaxed to arbitrary packet sizes providing that the buffer is always allocated as an integer number of packet size and the network uses store-and-forward or virtual cut-through so that each packet can be treated as an atomic entity.

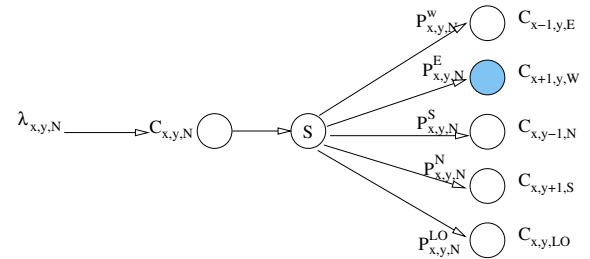


Figure 6: Queuing model of a channel

As shown in Fig. 6, we derive the queue model as observed by an input channel (in our example  $C_{x,y,N}$ ) by separating out the remaining part into two distinct queues. The bubble labeled with  $S$

<sup>4</sup>To make the figure more readable, not all the connections and parameters are explicitly shown in Fig. 5.

models the delay involved in router's service delay (Eq. (1)), while the five bubbles in the right hand side model the delay of the packet to be accepted by each downstream input channels (Fig. 6).

Now let us consider the behavior of a downstream channel and use  $C_{x+1,y,W}$  as an example. It can accept a new packet in the next clock cycle provided that it still has available space in its FIFO, while no packet can be accepted when its FIFO is full. Thus, we can use the reciprocal of the blocking probability to approximate the service rate that can be provided to the upstream router. More specifically, the *effective service rate* as observed by  $C_{x,y,N}$  is approximated as  $\frac{1}{b_{x+1,y,W}}$ . Since the total arrival rate to channel  $C_{x+1,y,W}$  is  $\lambda_{x+1,y,W}$ , by using *Little's Formula* [13], the average waiting time for entering the FIFO of  $C_{x+1,y,W}$  can be approximated as:

$$w_{x+1,y,W} = \frac{1}{\frac{1}{b_{x+1,y,W}} - \lambda_{x+1,y,W}} \quad (6)$$

Note that the contention delay over the link between on-chip routers is also taken into consideration in Eq. (6). On the other hand,  $w_{x+1,y,W}$  should also be the average waiting time observed by a packet in the channel  $C_{x,y,N}$  if it is to be delivered eastward to  $C_{x+1,y,W}$ . In order to facilitate the analysis, we now assume that  $C_{x,y,N}$  has an equivalent separate eastward queue without competing with other input channels. The arrival rate of this queue is then  $p_{x,y,N}^E \times \lambda_{x,y,N}$ . If this virtual queue should provide the same average latency packet, then its service rate  $\bar{\mu}_{x,y,N}^E$  must satisfy the following equation, again based on *Little's Formula*:

$$w_{x+1,y,W} = \frac{1}{\bar{\mu}_{x,y,N}^E - p_{x,y,N}^E \times \lambda_{x,y,N}} \quad (7)$$

Substituting  $w_{x+1,y,W}$  in Eq. (6) with the right hand side of Eq. (7), we can calculate the equivalent service rate of this virtual queue ( $\bar{\mu}_{x,y,N}^E$ ) by:

$$\bar{\mu}_{x,y,N}^E = \frac{1}{b_{x+1,y,W} - \lambda_{x+1,y,W} + p_{x,y,N}^E \times \lambda_{x,y,N}} \quad (8)$$

The average service contributed by all five downstream channels can now be calculated by the following equation:

$$\bar{\mu}_{x,y,N} = p_{x,y,N}^N \times \bar{\mu}_{x,y,N}^N + p_{x,y,N}^E \times \bar{\mu}_{x,y,N}^E + p_{x,y,N}^W \times \bar{\mu}_{x,y,N}^W + p_{x,y,N}^S \times \bar{\mu}_{x,y,N}^S + p_{x,y,N}^{LO} \times \bar{\mu}_{x,y,N}^{LO} \quad (9)$$

With this representation of  $\bar{\mu}_{x,y,N}$ , the model in Fig. 6 can be further reduced as shown in Fig. 7.

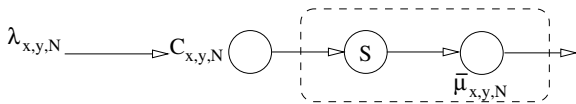


Figure 7: Reduced model of a channel

The next step to further simplify the model is merging the two queues as shown in the dashed box in Fig. 7<sup>5</sup>. Thus, the average queue length of  $C_{x,y,N}$  can be approximated by:

$$q_{x,y,N} = \frac{1}{\mu_{x,y,N} - \lambda_{x,y,N}} \quad (10)$$

On the other hand, if we treat the two queues in the dashed box as two independent M/M/1 queues, then the average queue length

<sup>5</sup>Think of the dashed box in Fig. 7 to act as a server and serving the packet in  $C_{x,y,N}$  with the rate of  $\mu_{x,y,N}$ .

of  $C_{x,y,N}$  should be equal to the sum of the length of these two queues:

$$q_{x,y,N} = \frac{1}{1/S - \lambda_{x,y,N}} + \frac{1}{\bar{\mu}_{x,y,N} - \lambda_{x,y,N}} \quad (11)$$

Combining Eq. (10) and Eq. (11) together, we have:

$$\mu_{x,y,N} = \lambda_{x,y,N} + \frac{1}{\frac{1}{1/S - \lambda_{x,y,N}} + \frac{1}{\bar{\mu}_{x,y,N} - \lambda_{x,y,N}}} \quad (12)$$

At this point, we have described the relation between the channel service rate  $\mu$  and the channel blocking probability  $b$  (by combining Eqs. (8), (9) and (12)). By performing similar derivations for all input channels, we can finally build a series of equations which describe the system's behavior. When given other parameters (i.e., routing function  $\mathcal{R}(j, k, j', k', x, y, dir)$ ,  $a_{x,y}$ ,  $d_{x,y}^{x',y'}$ ,  $S$  and  $l_{x,y,dir}$ ), these equations can be solved together by a nonlinear equation solver to determine the important parameters related to the system performance (such as  $\mu_{x,y,dir}$  and  $b_{x,y,dir}$ ). This is described next.

## 4.2 The buffer allocation algorithm

We propose an efficient greedy algorithm to solve this problem based on the aforementioned analytical model. The flow of algorithm is shown in Fig. 8.

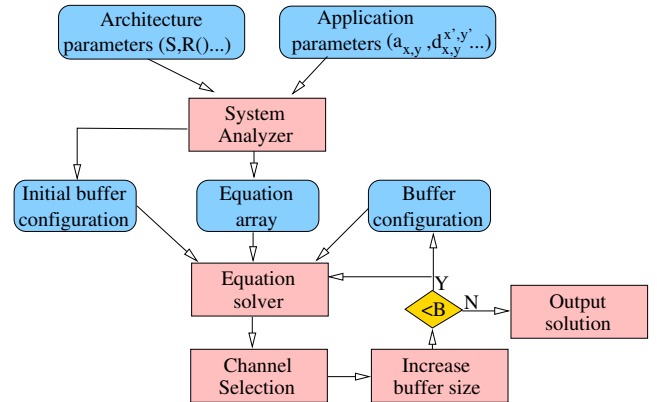


Figure 8: The buffer allocation algorithm flow

Given the architecture parameters (such as the routing algorithm, delay parameters, etc.), and the application parameters ( $a_{x,y}$  and  $d_{x,y}^{x',y'}$ ), the *System Analyzer* (written in C++) automatically generates the system equations for all the routers using the above modeling technique and writes the equations to a *Matlab* script file. At the same time, it also generates the initial buffer configuration which assigns the buffer in all the used channels ( $\lambda_{x,y,dir} \neq 0$ ) to be one packet large. Next, the *Equation Solver* is used to solve the given equations and determine  $b_{x,y,dir}$  for each input channel. Currently, in our tool flow, the *fsolve* utility from *Matlab* is used as the nonlinear solver.

In the next step, the channel which has the largest  $b_{x,y,dir}$  is selected as the bottleneck channel and the size of its buffer ( $l_{x,y,dir}$ ) is incremented by one packet. The above procedure is repeated until the total buffering space used in all the channels across the chip reaches the buffer limit  $B$  (i.e.  $\sum_{\forall x} \sum_{\forall y} \sum_{\forall dir} l_{x,y,dir} = B$ ).

Let  $C$  to be the number of channels participating in channel buffer allocation and  $F(C)$  to be the complexity of the nonlinear

solver used in solving the equations. Then the complexity of the buffer allocation algorithm is  $O(B \times F(C))$ , as it will invoke the solver for  $O(B)$  iterations. Although simple in nature, the algorithm performs extremely well in allocating buffering resources, as demonstrated by the experimental results.

## 5. EXPERIMENTAL RESULTS

### 5.1 Evaluations under random traffic

In this set of experiments, we applied our algorithm to applications with random traffic models [6][12]. In addition, we also tested our algorithm with different routing schemes, as well as different buffering resource budget values ( $B$ ).

Table 2 shows the average packet latency comparison between the NoCs with uniformly allocated FIFO buffers (denoted by *UNoC* hereafter) and the systems that benefited from customized buffer allocation (denoted by *CNoC*). All the NoCs reported in Table 2 are composed of  $4 \times 4$  tiles and use *XY* routing. (In short, for *2D* mesh networks, the *XY* routing first routes packets along the *X*-axis. Once the packets reach the column where lies the destination tile, they are then routed along the *Y*-axis.) Two traffic patterns<sup>6</sup> used in this evaluation are *uniform* and *hot spot*. Under the *uniform* traffic pattern, a PE sends a packet to any other node with equal probability. Under *hot spot* traffic pattern, one or more nodes are chosen as *hot spots* which receive an extra proportion of traffic in addition to the regular uniform traffic.

**Table 2: Packet latency (cycles) comparison (XY routing)**

| Pattern           | <i>UNoC</i> (B=96) | <i>UNoC</i> (B=144) | <i>CNoC</i> (B=96) |
|-------------------|--------------------|---------------------|--------------------|
| <i>uniform</i>    | 2877.64            | 104.11              | 2877.64            |
| <i>1hotspot-1</i> | 1538.01            | 716.96              | 1161.91            |
| <i>1hotspot-2</i> | 1388.44            | 99.17               | 371.78             |
| <i>2hotspot</i>   | 1656.64            | 100.61              | 144.77             |
| <i>3hotspot</i>   | 1335.70            | 74.06               | 145.94             |

Referring to Table 2, both *1hotspot-1* and *1hotspot-2* have only one hot spot, which is located at  $PE_{2,2}$  and  $PE_{0,1}$ , respectively. *2hotspot* and *3hotspot* are the *hot spot* traffic patterns which have two and three hot spots, respectively, with the hot spots' location arbitrarily selected across the chip. For each traffic pattern, we set the packet injection rate such that the system with uniform FIFO allocation works close to its critical point (that is, close to the bending point in Fig. 3).

The second column in Table 2 shows the average packet latency of the *UNoC* where each input channel has a FIFO size of 2 packets (thus the total buffering resource used is  $B = 96$ , that is, 48 links times 2), while each number in the last column (*CNoC*) shows the performance of an NoC customized under the corresponding traffic pattern using the exact same amount of 96 total buffering space. As we can see, except for the *uniform* traffic pattern, significant performance improvements are observed by allocating the buffer sizes according to the corresponding application traffic pattern.

For *uniform* traffic pattern, our allocation algorithm distributes the buffering space uniformly across all the input channels. The reason for this interesting result is that under *uniform* traffic, the *XY* routing happens to spread the packets almost evenly across the mesh. Since the total buffering space is very tight (on average each channel only gets a buffer size of 2), allocating the buffering space uniformly appears to best match the traffic pattern. Also, reported

<sup>6</sup>The evaluation results using other network sizes and other random traffic patterns are consistent but not reported here due to space limitations.

in the third column of Table 2 is the average packet latency of the *UNoC* where each input channel has a FIFO size of 3 ( $B = 144 = 48 \times 3$ ). Under all these traffic patterns, *UNoC* with  $B = 144$  always performs better than *CNoC* with  $B = 96$ . Please note that *UNoC* with  $B = 144$  requires 50% more buffering space compared to *CNoC* with  $B = 96$ .

To see the impact of the total available buffering space ( $B$ ), we applied the algorithm with the limit of total buffering space  $B = 192$ . Thus, each input channel in the corresponding *UNoC* has a size of 4; the results are shown in Table 3. Compared to Table 2, the packet injection rate has been increased in order to make the optimization really necessary and the results more interesting.

**Table 3: Packet latency (cycles) comparison (XY routing)**

| Pattern           | <i>UNoC</i> (B=192) | <i>UNoC</i> (B=240) | <i>CNoC</i> (B=192) |
|-------------------|---------------------|---------------------|---------------------|
| <i>uniform</i>    | 1812.52             | 705.75              | 998.18              |
| <i>1hotspot-1</i> | 1208.34             | 1106.33             | 881.18              |
| <i>1hotspot-2</i> | 1346.6              | 870.62              | 63.24               |
| <i>2hotspot</i>   | 1377.14             | 469.54              | 133.66              |
| <i>3hotspot</i>   | 1464.48             | 740.10              | 90.86               |

Again, in this case, *CNoC* performs much better than *UNoC* ( $B = 192$ ) under the same buffering constraints. Moreover, the increase in the buffering space budget offers more flexibility and finer control in buffer allocation, which enables the generation of better configurations even for the *uniform* traffic pattern. In fact, except for the *uniform* traffic, *CNoC* performs even better than *UNoC* ( $B = 240$ ). This means that by customizing the FIFO size according to the traffic pattern, the proposed algorithm is able to generate systems with much better performance while using 20% less buffering resources compared to systems with uniformly assigned FIFO sizes.

To show that our algorithm can be used for NoCs with other deterministic routing algorithms as well, we applied it to NoCs with *Odd-even fixed* (OE-fixed) routing. *OE-fixed* is indeed a deterministic version of *odd-even* [6] routing by removing the *odd-even*'s adaptiveness. For instance, in *odd-even* routing, if a packet with a given source and destination can be routed to both direction  $dir_1$  and  $dir_2$ , it will always be routed to  $dir_1$  in *OE-fixed*.

**Table 4: Packet latency (cycles) comparison (OE-fixed routing)**

| Pattern           | <i>UNoC</i> (B=96) | <i>UNoC</i> (B=144) | <i>CNoC</i> (B=96) |
|-------------------|--------------------|---------------------|--------------------|
| <i>uniform</i>    | 1176.15            | 262.64              | 182.05             |
| <i>1hotspot-1</i> | 1373.46            | 590.90              | 799.85             |
| <i>1hotspot-2</i> | 2195.54            | 628.12              | 685.32             |
| <i>2hotspot</i>   | 2119.42            | 594.42              | 728.13             |
| <i>3hotspot</i>   | 1359.22            | 795.96              | 873.55             |

As we can see, from Table 4, significant performance improvement is again achieved by performing application-specific buffer size customization. It is interesting to note that, unlike in *XY* routing, in this case *CNoC* performs much better compared to *UNoC* ( $B = 96$ ) under *uniform* traffic. The reason is that *OE-fixed* routing does not spread the traffic evenly across the mesh under *uniform* pattern, which makes uniform buffer allocation unsuitable.

Finally, we would like to point out that the proposed algorithm is also very fast. Take the buffer allocation for  $4 \times 4$  NoC with *XY* routing and  $B = 192$  as an example (this corresponds to the data in the second and fourth columns in Table 3), the *run time for generating the buffer allocation* for each traffic pattern is less than 100 seconds, when running on a desktop with a Pentium III 1 GHz processor.

## 5.2 Evaluations under realistic traffic conditions

We also evaluated the performance of our algorithm by applying it to applications which mimic real world traffic. Three benchmark applications are collected, namely *auto-indust*, *telecom* and *audio-video*. Both *auto-indust* and *telecom* are retrieved from *E3S* benchmark suites [10], which contain 24 and 30 tasks, respectively. The *audio-video* benchmark includes a video encoder/decoder pair and an audio encoder/decoder with a total of 40 tasks. For each benchmark, we manually assigned its tasks onto a  $4 \times 4$  NoC. The communication rates between any two tiles is set to be proportional to the communication volume between these two tiles. In our analysis and simulation, packets are generated with exponential distributions which may not always be true in reality. However, since the packet rates between different tiles are assigned to be proportional to the total communication volume between the two tiles, this model still has a good value in characterizing the heterogeneous traffic nature for the chosen benchmarks.

We customized next the NoC for each benchmark using a buffer budget  $B = 96$ ; the performance comparison is shown in Table 5. *XY* routing is assumed in all these applications.

**Table 5: Packet latency (cycles) comparison**

| Benchmark          | UNoC (B=96) | UNoC (B=144) | CNoC (B=96) |
|--------------------|-------------|--------------|-------------|
| <i>auto-indust</i> | 701.73      | 274.049      | 35.68       |
| <i>telecom</i>     | 1826.63     | 1137.23      | 68.91       |
| <i>audio-video</i> | 908.29      | 573.37       | 181.59      |

Comparing Table 5 with Table 2, it is clear that the buffer allocation is even more beneficial for these benchmarks. Indeed, compared to the random traffic patterns in Table 2, the traffic patterns for these benchmarks are much more unbalanced (for instance, some of the channels have even a load of zero), which makes the idea of application-specific buffer allocation more attractive. As we can see, *CNoC* (B=96) performs better than *UNoC* (B=144) in all these cases. In fact, in order to achieve an equivalent or shorter packet latency as *CNoC* (B=96), the average buffer length for a *UNoC* should be increased to 27, 9 and 14, respectively. This means that, in order to achieve the same performance as a NoC with customized buffer allocation, the NoC implementation with uniformly distributed buffer size will need to consume 12.5, 3.5 and 6 times more buffering space.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we have shown that, in order to minimize the implementation costs and maximize the NoC performance, the buffering size in each channel has to be carefully allocated in order to match the application's traffic pattern. An efficient greedy algorithm was proposed, which automatically allocates the buffering resources to different NoC channels, such that the communication performance is maximized while satisfying the total buffering resource budget.

The choice of using a *2D* mesh network as the underlying NoC architecture serves only as an example. Indeed, our algorithm can be extended to arbitrary topologies by adapting the analytical models in Section 4.1 accordingly to the target topology. Moreover, although we have evaluated our algorithm only for NoCs with *XY* and *OE-fixed* routing, the approach is general enough to be applied to any deterministic routing. In fact, the approach can even be applied to any *oblivious* routing [22], since in this case the arrival rate of each channel can still be calculated as shown in Eq. (3).

We plan to advance this research in several directions. One possible extension is to extend this approach for NoCs with adaptive routing. The main obstacle is the difficulty involved in the calcu-

lation of the arrival rate for each channel as multiple routing paths are possible in adaptive routing. Another important extension is to support the NoC with wormhole routing. In this case, a new analytical model needs to be derived, as it is necessary to treat each flit (instead of a packet) as a separate customer. Moreover, the header flit and the payload flits can no longer be treated as independent as they are tightly coupled. Finally, we are working on FPGA prototype and plan to use it for accurate evaluation of the effectiveness of the buffer allocation algorithm.

## References

- [1] A. Hemani, *et al.* Network on a chip: an architecture for billion transistor era. In *Proc. of the IEEE NorChip Conf.*, Nov. 2000.
- [2] A. Jalabert, *et al.* xPipesCompiler: a tool for instantiating application-specific NoCs. In *Proc. DATE*, Feb. 2004.
- [3] V. S. Adve and M. K. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Tran. on Parallel and Distributed Systems*, 5:225–246, March 1994.
- [4] A. Agarwal. Limits on interconnection network performance. *IEEE Tran. on Parallel and Distributed Systems*, 2(4):398–412, Oct. 1991.
- [5] L. Benini and G. De Micheli. Networks on chip: A new paradigm for systems on chip design. In *Proc. DATE*, March 2002.
- [6] G.-M. Chiu. The odd-even turn model for adaptive routing. *IEEE Tran. on Parallel and Distributed Systems*, 11(7):729–738, July 2000.
- [7] W. J. Dally. Performance analysis of *k*-ary *n*-cube interconnection networks. *IEEE Tran. on Computers*, 39(6):775–785, June 1990.
- [8] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1(3):187–196, 1986.
- [9] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. DAC*, June 2001.
- [10] R. Dick. Embedded system synthesis benchmarks suites (E3S). <http://helsinki.ee.princeton.edu/~dickrp/e3s/>.
- [11] E. Bolotin, *et al.* QNoC: QoS architecture and design process for network on chip. *Special issue on Networks on Chip, The Journal of Systems Architecture*, Dec. 2003.
- [12] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *Journal of ACM*, 41(5):874–902, Sept. 1994.
- [13] F. S. Hillier and G. J. Lieberman. *Introduction to operations research*, chapter 15, pp. 661–732. McGraw-Hill, sixth edition, 1995.
- [14] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proc. DATE*, March 2003.
- [15] J. Hu and R. Marculescu. Smart routing for networks-on-chip. Technical report, ECE Department, Carnegie Mellon University, March 2004. Available at <http://www.ece.cmu.edu/~sld/pubs>.
- [16] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. In *Computer Networks*, volume 3, pp. 267–286, Sept. 1979.
- [17] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. DATE*, Feb. 2004.
- [18] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Tran. on Computers*, 26:62–76, Feb. 1993.
- [19] L.-S. Peh and W. J. Dally. A delay model for router micro-architectures. *IEEE Micro*, 21(1):26–34, Jan.–Feb. 2001.
- [20] S. Kumar *et al.* A network on chip architecture and design methodology. In *Proc. Symposium on VLSI*, April 2002.
- [21] Semiconductor Association. *The International Technology Roadmap for Semiconductors (ITRS)*, 2001.
- [22] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal of Computing*, 11(2):350–361, May 1982.