System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design

Jingcao Hu, *Member, IEEE*, Umit Y. Ogras, *Student Member, IEEE*, and Radu Marculescu, *Member, IEEE*

***Abstract*—In this paper, a novel system-level buffer planning algorithm that can be used to customize the router design in networks-on-chip (NoCs) is presented. More precisely, given the traffic characteristics of the target application and the total budget of the available buffering space, the proposed algorithm automatically assigns the buffer depth for each input channel, in different routers across the chip, 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. Indeed, the experimental results show that while the proposed algorithm is very fast, significant performance improvements can be achieved compared to the uniform buffer allocation. For instance, for a complex audio/video application, about 80% savings in buffering resources, can be achieved by smart buffer allocation using the proposed algorithm.**

***Index Terms*—Buffer sizing, design automation, low power, networks-on-chip (NoCs), optimization.**

I. Introduction

With the recent advances in the semiconductor technology, it is possible for designers to integrate on a single chip tens of Intellectual Property (IP) blocks together with large amounts of embedded memory. 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 feature size in the deep submicrometer (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, electromagnetic interference, etc. [31].

The NoC approach was proposed as a promising solution to these complex on-chip communication problems [4], [10], [15], [21]. For the NoC 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 an NoC implementation where nodes are connected using a simple two-dimensional (2-D) 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 2-D mesh NoC is shown in Fig. 1(b)]. As such, instead of routing design-specific global wires, the internode 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

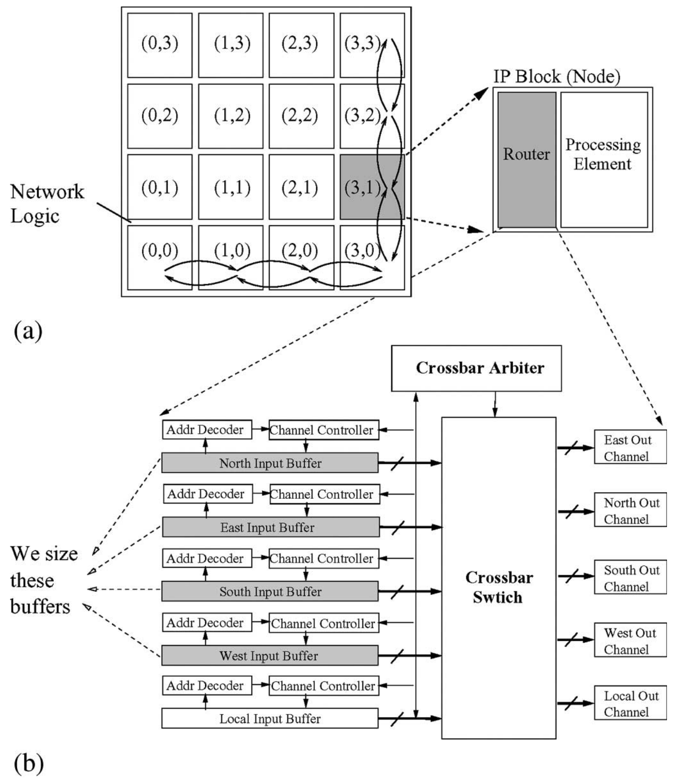


Fig. 1. (a) NoC implementing a 2-D mesh topology. (b) Typical on-chip router architecture.

Manuscript received March 1, 2005; revised July 5, 2005 and November 23, 2005. This work was supported in part by the National Science Foundation (NSF) under Grant CCR-00-93104 and in part by Marco Gigascale Systems Research Center (GSRC). This paper was recommended by Associate Editor R. Gupta.

J. Hu was with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890 USA. He is now with Tabula, Inc., Santa Clara, CA 95054 USA (e-mail: jhu@tabula.com).

U. Y. Ogras and R. Marculescu are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890 USA (e-mail: uogras@ece.cmu.edu; radum@ece.cmu.edu).

Digital Object Identifier 10.1109/TCAD.2006.882474

the silicon area of the NoC [18], [30]; consequently, 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 vary significantly across different applications, the buffering resources have to be judiciously allocated to each input channel 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.

1) We propose an efficient algorithm that 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 communicating IP pairs, and other relevant architectural parameters (e.g., routing algorithm and arbitration delay), 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 for 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 units while keeping the size of the "south input buffer" in the router located at tile (2,2) to be one unit. This buffer sizing can be done based on 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 around 80% less buffering resources. Besides area savings, this reduction in buffer size is also important from a power dissipation perspective.
2) We propose a novel analytical model that can be used to quickly analyze and detect the potential performance bottlenecks in different router channels. This is accomplished 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 of 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 II, we give a brief review of the relevant work. The problem of buffer allocation for NoCs is then described in Section III. Following that, in Section IV, an efficient heuristic is proposed to solve this problem. The basic idea is to iteratively add more buffering resources to the router channel identified as being the performance bottleneck. Experimental results in Section V show that significant performance improvements can be achieved while satisfying the same total buffering space budget, compared to uniform buffer allocation. Finally, we summarize our contribution and suggest directions for future work.

II. 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 to achieve best energy, performance, and cost tradeoffs [4], [22]. In [19] and [27], the authors show the benefits of the network topology customization on the system area, power, and latency. The authors of [17] and [24] 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 in providing an efficient way to solve the buffer space allocation problem for NoC designs. Traditional work on performance evaluation in parallel computing uses either time-consuming simulation (e.g., [7] and [14]) or provides analytical models for limited traffic conditions (typically uniform models) [2], [8]. 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 [1] model a wormhole-based [9] 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, these analytical models apply only to networks with infinite buffers (e.g., [2] and [8]) and/or single-flit buffers [1]. As such, they cannot be used for buffer allocation in which the effect of arbitrary, but finite, buffer size has to be explicitly modeled.

The network calculus method [5] introduced by Le Boudec and Thiran uses max–min algebra to derive some useful properties of macro networks. Particularly related to this paper, it can be used to compute the minimal buffer size required given a maximal peak rate or even some more complex arrival traffic pattern. However, one main problem in applying the network calculus to on-chip network is that the bound derived by the network calculus can be very loose. Although this may not be a serious problem for macro networks (which typically have much larger buffer sizes), this limitation makes network calculus not suitable for on-chip networks in which the buffer sizes are typically very small.

From a different perspective, the design issue of efficient buffering structure for on-chip NoC has been addressed [13], [30]. In [13], custom-made hardware first-in-first-out (FIFO) is used in favor of RAMs due to performance and area considerations. In [30], the implementation structure of the buffer for Proteo NoCs is presented. However, neither of them addressed the problem of customizing buffer sizes to match the application traffic under consideration.

Chandra *et al.* [6] investigated the impact of interconnect FIFO sizing on interconnect throughput. However, the research in that paper focuses on single-source single-sink interconnect, thus cannot be directly applied to the buffer size allocation problem addressed in this paper, as the impacts of traffic merging, splitting, etc., have to be considered. Moreover, the results provided in [6] are based on time-consuming simulation. Such an approach is not suitable in buffer size allocation problem addressed in this paper as the solution space that we need to explore explodes as a function of problem size.

III. Problem of Router Buffer Allocation for NoCs

Simply stated, given the communication probability profile between each communicating pair of IPs 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 that are relevant to our algorithm. We then formalize the problem and illustrate the significance of finding a good solution to it.

A. System Characterization

1) Platform Characterization: The system under consideration is composed of $n \times n$ tiles interconnected by a 2-D mesh network. Because of its simplicity, the choice of a 2-D mesh as the underlying NoC architecture serves only as an example for our algorithm.[1] We further assume that deterministic routing (e.g., dimension-ordered routing [25]) 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 [20] 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:

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

In (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. Note that S in (1) represents the delay of a packet being transmitted across the router when there are no other packets present (i.e., no congestion). To calculate the real network delay, queuing/blocking delay should be also included; this will be further described in Section IV-A. We also note that different router architectures may lead to different delay models (e.g., [28]), 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) that buffers 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" to set up a path to the corresponding output channel.

The actual use of the input buffer is regulated through a backpressure 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 $m \times \mathrm{SP}$, where m is a positive integer. We also assume the size of the "local input" buffer [the input channel that accepts packets from the router's local processing element (PE), see Fig. 1(a)] to be infinite. As discussed later in this paper, 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 explicitly 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 controller requests for the same available

[1]The algorithm can be extended for arbitrary topologies and sizes as it will be explained later.

TABLE I
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	
dir	Direction, *i.e. North, South, East, West, Local*	
$\mathcal{R}$	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 $C_{x,y,dir}$	
$a_{x,y}$	Packet injection rate of $PE_{x,y}$	*Application Specific Parameters*
$d_{x,y}^{x',y'}$	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	

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.

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 PE injects packets with a “Poisson” distribution. More specifically, let (x, y) be the location of the tile at the intersection of 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 to provide an analytical solution to the buffer allocation problem.

B. Problem Formulation

For convenience, we summarize the basic parameters in Table I.[2] With these notations, the problem of router buffer allocation for 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 $l_{x,y,\mathrm{dir}}$ for each input channel that minimizes the average packet latency L, i.e.,

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

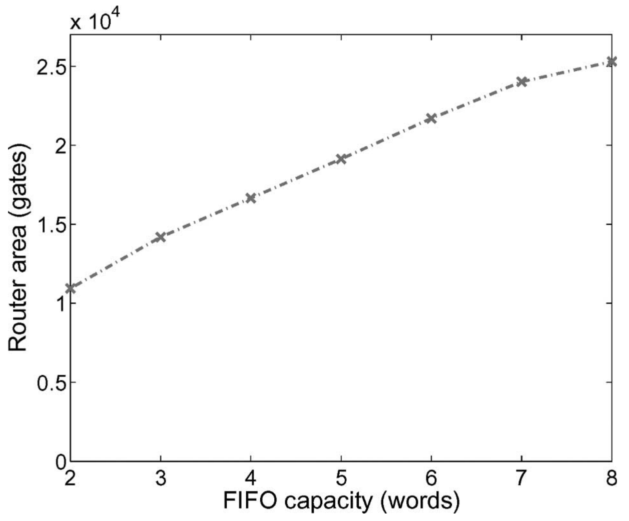


Fig. 2. Router sizes with different FIFO capacity per input channel. X-axis gives the FIFO capacity for each input channel. Y-axis gives the router size in number equivalent gates.

C. Significance of the Problem

To reduce the NoC implementation cost and power dissipation, small on-chip routers are definitely needed. To better understand the main consumers of resources, we prototyped an on-chip router based on the architecture shown in Fig. 1(b) [18]. The router supports XY routing with FCFS crossbar arbiter and uses a 0.16-μm 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.[3]

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), whereas 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 two to three words leads to an increase of total router area by 30%.

The layout of the router with each channel having uniformly assigned an FIFO size of four words is provided in Fig. 3, where

[2]Some of the parameters will be explained later in more detail.

[3]More specifically, the IP DW_fifo_s2_sf from Synopsys DesignWare Foundation library is instantiated as our FIFO modules in this design.

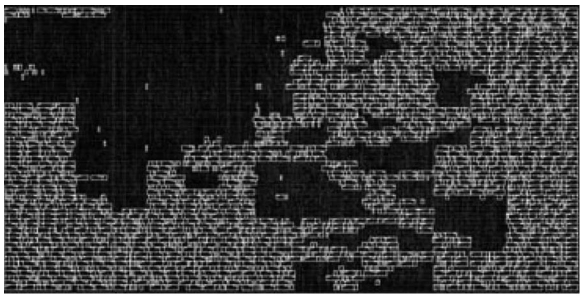


Fig. 3. Router layout with FIFO capacity of four words. Area taken by FIFO is highlighted in yellow (light) color.

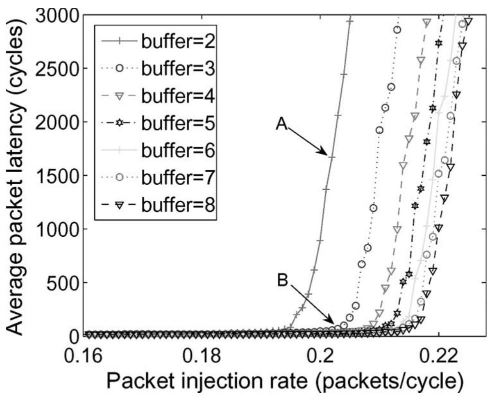


Fig. 4. System performance with different FIFO capacities.

the area occupied by FIFOs is highlighted. As expected, the FIFO takes a significant part of the total area, as has also been observed by other researchers (e.g., [29]). Thus, to minimize the implementation overhead of NoC, an effective approach is to reduce the overall use of the buffering space in the routers.

In general, the more buffering resources become available, the better the performance gets. To show the impact of the FIFO capacity on the communication performance, we simulated a 4×4 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 are collected after chunks of 20 000 packets are sent. A cycle-accurate interconnection network simulator ("Nets") was implemented in C++. Nets supports 2-D mesh networks with packet routing; it has been designed to be easily customized to simulate different designs under various traffic patterns. Since many factors (e.g., routing path selection delay and crossbar arbitration delay) have a significant impact on the NoC performance, Nets models them accurately with the actual values taken from the prototype router design.

Fig. 4 shows a typical latency/throughput plot under different communication loads and FIFO sizes. The simulated traffic pattern is "hot spot,"[4] where the PEs located at tiles (1,0) and (2,2) [see Fig. 1(a)] are the ones chosen as hot spots. The impact of the buffer size on the average packet latency is obvious from Fig. 4. 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 0.203 packets per clock cycle, the average packet latency of the system with a uniform buffer size of two is 2061 clock cycles (point A in Fig. 4), whereas the average packet latency for a uniform buffer size of three is only 68 clock cycles (point B in Fig. 4).

[4]The hot spot traffic is explained in Section V.

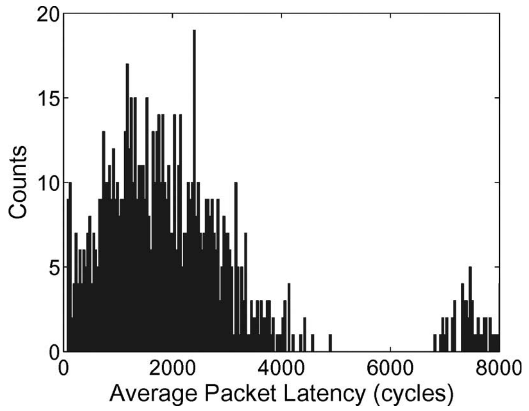


Fig. 5. Histogram for 1000 different random buffer configurations.

On the other hand, increasing "uniformly" the capacity of every FIFO in every on-chip router may not be the most effective way 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 $4 \times \mathrm{SP}$ 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 $4 \times \mathrm{SP} \times 48 = 192 \times \mathrm{SP}$ (i.e., $B = 192$). We randomly generate 1000 solutions, all of them using a size of $192 \times \mathrm{SP}$ buffering space. Each configuration is then simulated under the injection rate of 0.212 packets/s. The performance is shown as a histogram in Fig. 5.

As we can see, the solutions vary significantly in terms of average packet latency, despite the fact that all of them use exactly the 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 66 clock cycles, which is significantly better than the performance of the system having every buffer uniformly assigned of size $4 \times \mathrm{SP}$, whose average packet latency is as high as 614 clock cycles. (There is a factor of 10 difference between the two latency values.)

On the other hand, 1000 random solutions represent only a small portion of the entire solution 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 33 clock cycles at the injection rate of

0.212 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 × SP, which has an average packet latency of 39 clock cycles at this injection rate. Consequently, by customizing the buffer size, we can actually implement a system that has better performance but uses only half of the buffering resources compared to a system with FIFO size uniformly assigned to 8 × SP. This fully justifies the approach we take for FIFO size customization.

As a side note, we would like to point out that, compared to uniform buffer size allocation, we expect significant leakage energy savings as a by-product for the systems having the buffer sizes customized. The reason is that by allocating the buffer size according to the traffic pattern of the target application, the total buffering resources used in the system can be significantly reduced without degrading its performance. Since the total leakage power is proportional to the total buffering space, the system customized by buffer size allocation should thus consume less leakage power. However, since this paper focuses mainly on the buffer allocation problem for performance optimization, the energy issues are not addressed explicitly hereafter.

Obviously, the only way to find exactly the best solution is by enumerating and simulating all the possible solutions. Unfortunately, the solution space increases combinatorially with the network size and total buffer budget; therefore, the enumeration of the solution space is too expensive to afford, especially since we need to simulate each solution to evaluate the system performance. Thus, we propose next an efficient analytical heuristic.

IV. Solving the Router Buffer Allocation Problem

In this section, we present a novel buffer allocation algorithm that 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.

A. 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 that 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 that has the highest probability to be in the "full state." The channel that owns this particular FIFO becomes the real performance bottleneck in the current configuration, and thus, its size should be increased.

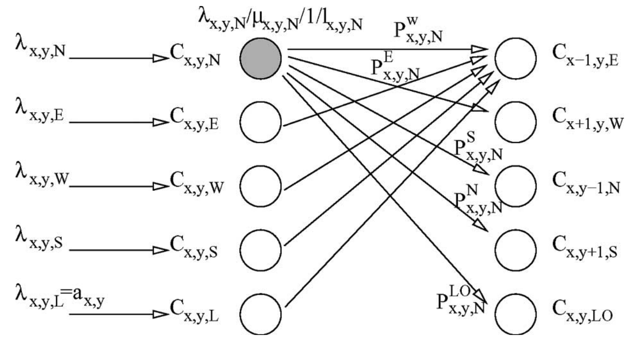


Fig. 6. Queuing model of a router.

We point out that to facilitate the analysis and make the problem manageable, the model needs to rely on some approximations. Nonetheless, as supported by our experimental results in Section V, the model we propose subsequently does provide an efficient solution for bottleneck channel identification.

To solve this problem analytically, we resort to the theory of finite queuing networks [16]. The basic element in the model is an 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 $C_{x,y,\mathrm{dir}}$ is modeled as a finite queue of length $l_{x,y,\mathrm{dir}}$, with the arrival rate $\lambda_{x,y,\mathrm{dir}}$, served by one server with service rate $\mu_{x,y,\mathrm{dir}}$. Both interarrival 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. 6. The five bubbles in the left-hand side represent the five channels of $R_{x,y}$ [i.e., 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 $R_{x,y}$.[5]

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. 6. Assuming the network is not overloaded (i.e., $\lambda_{x,y,\mathrm{dir}} < \mu_{x,y,\mathrm{dir}}$), then the arrival rate of $C_{x,y,N}$ can be calculated using the following equation:

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

[5]To make the figure more readable, only the most relevant connections and parameters are explicitly shown in Fig. 6.

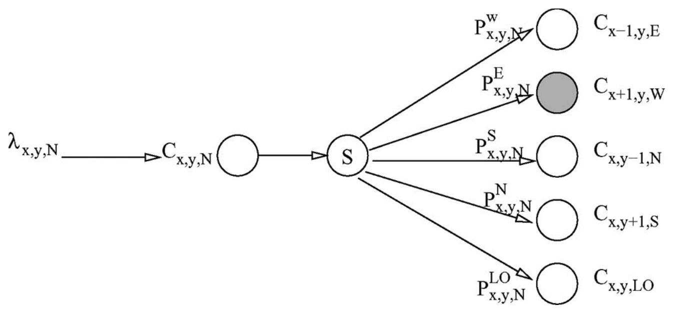


Fig. 7. Queuing model of a channel.

In (3), the routing function $\mathcal{R}(j,k,j',k',x,y,N)$ equals 1 if the packet from $\mathrm{PE}_{j,k}$ to $\mathrm{PE}_{j',k'}$ 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 $p^W_{x,y,N}$, $p^E_{x,y,N}$, $p^S_{x,y,N}$, $p^N_{x,y,N}$, and $p^{\mathrm{LO}}_{x,y,N}$ can be precalculated. (We use LO to represent the "local output" direction; thus, $p^{\mathrm{LO}}_{x,y,N}$ gives the probability of a packet in $C_{x,y,N}$ to be delivered to $\mathrm{PE}_{x,y}$).

Now, the only unknown parameter for channel $C_{x,y,N}$ is its service rate $\mu_{x,y,N}$. Once the value of $\mu_{x,y,N}$ is determined, the blocking probability of $C_{x,y,N}$ (i.e., the probability of $C_{x,y,N}$ to be in full state) can be calculated using the finite M/M/1/K queuing model with the following equations [16]:

$$\rho_{x,y,N} = \frac{\lambda_{x,y,N}}{\mu_{x,y,N}} \tag{4}$$

$$b_{x,y,N} = \frac{1-\rho_{x,y,N}}{1-\rho_{x,y,N}^{l_{x,y,N}+1}} \times \rho_{x,y,N}^{l_{x,y,N}}. \tag{5}$$

The calculation of $\mu_{x,y,N}$ is not trivial, as it depends not only on the router's service delay [as shown in (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 the west input channel of router $R_{x,y,N}$'s immediate east neighbor (i.e., $C_{x+1,y,W}$) is full, then the packet has to wait in channel $C_{x,y,N}$.

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 provided 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.

As shown in Fig. 7, we derive the queuing model 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 models the delay involved in router service delay (1), whereas the five bubbles on the right-hand side model the delay of the packet to be accepted by each downstream input channels (Fig. 7).

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 during the next clock cycle provided that it still has available

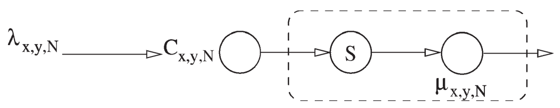


Fig. 8. Reduced model of a channel.

space in its FIFO, whereas 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" observed by $C_{x,y,N}$ is approximated as $1/b_{x+1,y,W}$. Since the total arrival rate to channel $C_{x+1,y,W}$ is $\lambda_{x+1,y,W}$, if we approximate the service provided by $C_{x+1,y,W}$ to its upstream router as an M/M/1 queue, the expected number of packets in the queue to be delivered to channel $C_{x+1,y,W}$ can then be calculated as [16]

$$N_{x+1,y,W} = \frac{\lambda_{x+1,y,W}}{\frac{1}{b_{x+1,y,W}} - \lambda_{x+1,y,W}}. \tag{6}$$

On the other hand, based on Little's formula [16]

$$N_{x+1,y,W} = \lambda_{x+1,y,W} \times w_{x+1,y,W}. \tag{7}$$

Now, replace $N_{x+1,y,W}$ in (6) with the right-hand side of (7). The average waiting time for entering the FIFO of $C_{x+1,y,W}$ can then be approximated as

$$w_{x+1,y,W} = \frac{1}{\frac{1}{b_{x+1,y,W}} - \lambda_{x+1,y,W}}. \tag{8}$$

Note that the contention delay over the link between the on-chip routers is also taken into consideration in (8). 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 $C_{x+1,y,W}$. 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 $p^E_{x,y,N} \times \lambda_{x,y,N}$. If this virtual queue should provide the same average latency packet, then its service rate $\bar{\mu}^E_{x,y,N}$ must satisfy the following equation, again based on Little's formula:

$$w_{x+1,y,W} = \frac{1}{\bar{\mu}^E_{x,y,N} - p^E_{x,y,N} \times \lambda_{x,y,N}}. \tag{9}$$

Substituting $w_{x+1,y,W}$ in (8) with the right-hand side of (9), we can calculate the equivalent service rate of this virtual queue $(\bar{\mu}^E_{x,y,N})$ by

$$\bar{\mu}^E_{x,y,N} = \frac{1}{b_{x+1,y,W}} - \lambda_{x+1,y,W} + p^E_{x,y,N} \times \lambda_{x,y,N}. \tag{10}$$

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

$$\begin{aligned}\bar{\mu}_{x,y,N} = {} & p^N_{x,y,N} \times \bar{\mu}^N_{x,y,N} + p^E_{x,y,N} \times \bar{\mu}^E_{x,y,N} + p^W_{x,y,N} \\ & \times \bar{\mu}^W_{x,y,N} + p^S_{x,y,N} \times \bar{\mu}^S_{x,y,N} + p^{LO}_{x,y,N} \times \bar{\mu}^{LO}_{x,y,N}.\end{aligned} \tag{11}$$

With this representation of $\bar{\mu}_{x,y,N}$, the model in Fig. 7 can be further reduced as shown in Fig. 8.

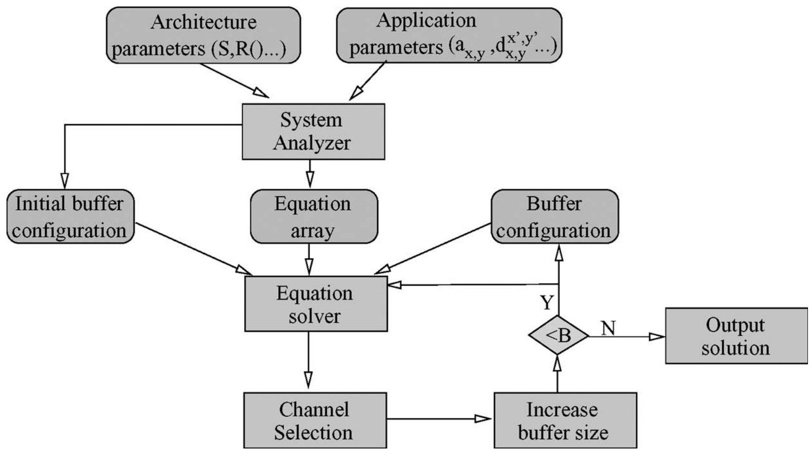


Fig. 9. Buffer allocation algorithm flow.

The next step to further simplify the model is merging the two queues as shown in the dashed box in Fig. 8.[6] Thus, the average queue length of $C_{x,y,N}$ can be approximated by

$$q_{x,y,N} = \frac{\lambda_{x,y,N}}{\mu_{x,y,N} - \lambda_{x,y,N}}. \tag{12}$$

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

$$q_{x,y,N} = \frac{\lambda_{x,y,N}}{1/S - \lambda_{x,y,N}} + \frac{\lambda_{x,y,N}}{\bar{\mu}_{x,y,N} - \lambda_{x,y,N}}. \tag{13}$$

Combining (12) and (13) together, we have

$$\mu_{x,y,N} = \lambda_{x,y,N} + \frac{1}{\frac{1}{1/S-\lambda_{x,y,N}} + \frac{1}{\bar{\mu}_{x,y,N}-\lambda_{x,y,N}}}. \tag{14}$$

At this point, we have described the relation between the channel service rate μ and the channel blocking probability b [by combining (10), (11), and (14)]. By performing similar derivations for all input channels, we can finally build a series of equations that describe the system's behavior. When given other parameters (i.e., routing function $\mathcal{R}(j,k,j',k',x,y,\text{dir})$, $a_{x,y}$, $d_{x,y}^{x',y'}$, S, and $l_{x,y,\text{dir}}$), these equations can be solved together by a nonlinear equation solver to determine the important parameters related to the system performance (such as $\mu_{x,y,\text{dir}}$ and $b_{x,y,\text{dir}}$). This is described next.

B. *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. 9.

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 that assigns the buffer in all the used channels $(\lambda_{x,y,\text{dir}} \neq 0)$ to be one packet large. Next, the "equation solver" is used to solve the given equations and determine $b_{x,y,\text{dir}}$ for each input channel. Currently, in our tool flow, the `fsolve` utility from `Matlab` is used as the nonlinear equation solver.

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

Let C be the number of channels participating in channel buffer allocation and $F(C)$ be the complexity of the nonlinear solver used in solving the equations. Then the theoretical "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.

Please note that, instead of the simple greedy algorithm introduced here, other more sophisticated algorithms (e.g., genetic algorithm, branch, and bound) can also used to solve the buffer allocation problem by leveraging our analytical model.

V. Experimental Results

A. *Evaluations Under Uniform and Nonuniform Random Traffic*

In this first set of experiments, we applied our algorithm to applications with random traffic models [7], [14]. In addition, we also tested our algorithm with different routing schemes, as well as different buffering resource budget values (B).

Table II shows the average packet latency comparison between the NoCs with uniformly allocated FIFO buffers (denoted by "UNoC" hereafter) and systems that benefit from customized buffer allocation (denoted by "CNoC"). All the

[6]Think of the dashed box in Fig. 8 as a server serving the packet in $C_{x,y,N}$ with the rate of $\mu_{x,y,N}$.

TABLE II
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (XY ROUTING, MESH TOPOLOGY, 4×4 NoC)

Pattern	***UNoC*** **(B=96)**	***UNoC*** **(B=144)**	***CNoC*** **(B=96)**
uniform	934.5	34.1	934.5
1hotspot-1	1026.1	191.6	698.5
1hotspot-2	734.7	41.8	79.1
2hotspot	892.1	45.0	94.0
3hotspot	1213.0	52.7	655.5

NoCs reported in Table II are composed of 4×4 tiles and use XY routing. (In short, for 2-D 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[7] 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 a hot spot traffic pattern, one or more nodes are chosen to receive an extra proportion of traffic in addition to the regular uniform traffic.

Referring to Table II, both "1hotspot-1" and "1hotspot-2" have only one hot spot, which is located at $\mathrm{PE}_{2,2}$ and $\mathrm{PE}_{0,1}$, respectively (see Fig. 1). "2hotspot" and "3hotspot" are the hot spot traffic patterns that 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 (i.e., close to the bending point in Fig. 4).

The second column in Table II shows the average packet latency of the UNoC where each input channel has a FIFO size of two packets (thus, the total buffering resource used is $B = 96$, i.e., 48 links times 2), whereas 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 a 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 two), allocating the buffering space uniformly appears to best match the traffic pattern. Also reported in the third column of Table II is the average packet latency of the UNoC where each input channel has a FIFO size of three ($B = 144 = 48 \times 3$). Under all these traffic patterns, UNoC with $B = 144$ always performs better than CNoC with $B = 96$. However, note that UNoC with $B = 144$ requires 50% more buffering space compared to CNoC with $B = 96$.

We also evaluated the gain of using our buffer allocation algorithm by applying it to a set of randomly generated benchmarks. Each of these six benchmarks (named *tgff0–tgff5* in Table III) is first generated using Task Graphs for Free (TGFF) [12] and then scheduled onto a 4×4 NoC architecture. The communication rates between different tiles are then estimated based on the scheduling results. The performance comparison between CNoC and UNoC are shown in Table III using the same notational convention of Table II.

[7]The evaluation results using other network sizes and other random traffic patterns are consistent but not reported here due to space limitations.

TABLE III
PACKET LATENCY (CYCLES) COMPARISON USING TGFF-GENERATED APPLICATIONS (XY ROUTING, MESH TOPOLOGY, 4×4 NoC)

Pattern	***UNoC*** **(B=96)**	***UNoC*** **(B=144)**	***CNoC*** **(B=96)**
tgff0	821.4	396.0	173.8
tgff1	636.5	225.4	87.1
tgff2	701.3	224.6	102.4
tgff3	837.6	109.5	37.1
tgff4	743.9	286.5	68.2
tgff5	609.7	73.4	21.4

TABLE IV
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (XY ROUTING, MESH TOPOLOGY, 4×4 NoC)

Pattern	***UNoC*** **(B=192)**	***UNoC*** **(B=240)**	***CNoC*** **(B=192)**
uniform	998.0	197.8	356.2
1hotspot-1	951.8	861.4	666.2
1hotspot-2	1026.9	402.5	36.7
2hotspot	1000.2	213.8	34.2
3hotspot	1053.9	571.3	116.4

Compared to Table II, Table III shows more improvement after applying the customized buffer allocation for these benchmarks. This is due to the fact that the traffic patterns used in Table III are more heterogeneous than the random traffic patterns in Table II. In fact, for all these test cases in Table III, the NoCs with customized buffer allocation (CNoC with $B = 96$) performs even better than NoCs with uniform buffer allocation (UNoC with $B = 144$), which uses 50% more buffering resources.

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 four; the results are shown in Table IV. Compared to Table II, the packet injection rate has been increased to make the optimization really necessary and the results more interesting.

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 a 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.

Fig. 10 shows how the buffer size in each router channel looks like after buffer allocation using the 1hotspot-1 traffic pattern as an example. Due to limited space, only the results for the north and south input channels of each router are shown

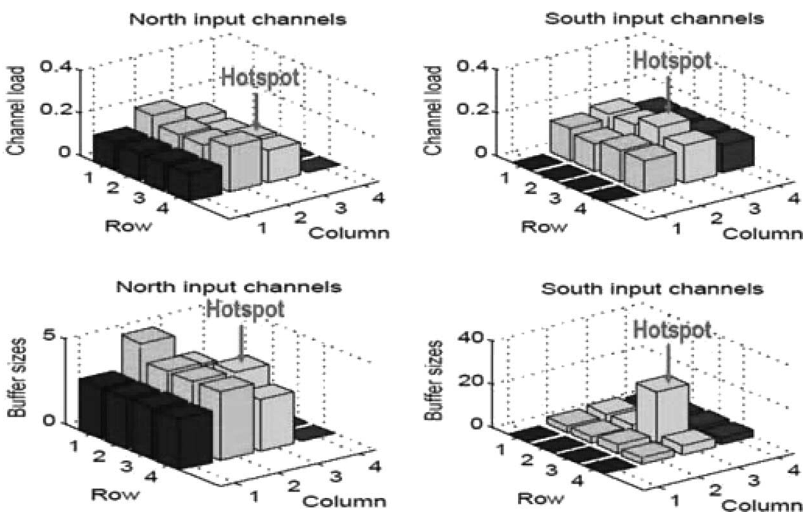


Fig. 10. Traffic load and buffer space distribution using hotspot traffic 1hotspot-1 as an example.

TABLE V
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (XY ROUTING, MESH TOPOLOGY, 6×6 NOC)

Pattern	*UNoC* (B=480)	*UNoC* (B=600)	*CNoC* (B=480)
uniform	1193.7	119.9	118.4
1hotspot-1	991.8	960.86	721.1
1hotspot-2	1320.9	665.4	34.1
2hotspot	1440.9	332.7	33.1
3hotspot	989.7	157.2	37.7

TABLE VI
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (XY ROUTING, TORUS TOPOLOGY, 4×4 NOC)

Pattern	*UNoC* (B=128, torus)	*UNoC* (B=192, torus)	*CNoC* (B=128, torus)
uniform	988.6	28.9	37.9
1hotspot-1	925.1	133.1	35.3
1hotspot-2	915.8	53.4	103.1
2hotspot	794.7	36.1	39.4
3hotspot	1046.7	76.7	50.8

in Fig. 10. The top half of Fig. 10 shows the traffic load of corresponding input channels, whereas the bottom half shows the allocated buffer sizes of the corresponding input channels. As we can see, the allocated buffer sizes vary significantly across different channels, and the correlation between the channel load and its allocated buffer size can be clearly identified.

To study the scalability effects, we also applied our buffer allocation algorithm to NoCs with different sizes. All the NoCs reported in Table V are composed of 6×6 tiles and use XY routing. Thus, the columns with $B = 480$ correspond to NoCs with an average input channel buffer size of four, whereas the column with $B = 600$ corresponds to NoCs with an average input channel buffer size of five. Again, a significant performance improvement is achieved by application-specific buffer allocation.

B. Evaluation for Other Topologies and Routing Schemes

As an example showing that the proposed algorithm can be applied to NoCs with arbitrary topologies, we applied it to NoCs implementing a "torus" topology. Reported in Table VI are the results of buffer allocation on 2-D 4×4 torus NoCs. Compared to the corresponding 4×4 mesh NoCs, the 4×4 torus NoCs have 16 extra links that are used to connect the boundary routers. Thus, the total buffering space budget (B in Table VI) is adjusted to be 128 for NoCs with an average FIFO size of two and 192 for NoCs with an average FIFO size of three, respectively. As shown in Table VI, a significant performance improvement is achieved through application-specific buffer size customization.

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 [7] routing. Odd–even routing belongs to the category of adaptive routing. To be deadlock free, an efficient adaptive routing needs to prohibit "at least one turn" in each of the possible routing cycles. In addition, it should not prohibit more turns than necessary to preserve the adaptiveness [14]. Proposed in [7], odd–even routing is developed by restricting the locations where some types of turns can take place such that the algorithm remains deadlock free. More precisely, the odd–even turn model is governed by the following two rules:

Rule 1. Any packet is not allowed to take an east-to-north turn at any nodes located in an even column, and it is not allowed to take a north-to-west turn at any modes located in an odd column.

Rule 2. Any packet is not allowed to taken an east-to-south turn at any nodes located in an even column, and it is not allowed to take a south-to-west turn at any nodes located in an odd column.

All the turns that satisfy the Rules 1 and 2 are considered legal in the odd–even routing; this leaves the router some adaptiveness in choosing the routing paths for a specific packet. More precisely, depending on the location and a packet's source and destination address, the packet can be routed to multiple directions.

TABLE VII
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (OE-FIXED ROUTING, MESH TOPOLOGY, 4 × 4 NOC)

Pattern	*UNoC* (B=96)	*UNoC* (B=144)	*CNoC* (B=96)
uniform	1037.8	119.9	78.5
1hotspot-1	1159.1	167.8	329.1
1hotspot-2	991.7	65.6	77.5
2hotspot	1113.0	50.9	484.0
3hotspot	1121.6	407.7	670.2

TABLE VIII
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (OE-SPLIT ROUTING, MESH TOPOLOGY, 4 × 4 NOC)

Pattern	*UNoC* (B=96)	*UNoC* (B=144)	*CNoC* (B=96)
uniform	877.0	31.0	68.6
1hotspot-1	978.3	36.4	49.6
1hotspot-2	853.2	35.5	43.1
2hotspot	851.1	32.9	38.8
3hotspot	830.8	30.4	42.3

The deterministic OE-fixed routing is developed by removing the minimum odd–even's adaptiveness. For instance, in odd–even routing, if a packet with a given source and destination can be routed to both directions dir_1 and dir_2, it will always be routed to dir_1 in OE-fixed. Obviously, since the turns allowed in OE-fixed are a subset of odd–even, OE-fixed is thus also free from deadlock.

As we can see from Table VII, 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.

Both of the above routing algorithms (i.e., XY and OE-fixed) belong to deterministic routing, which is indeed a special form of "oblivious" routing [32]. In oblivious routing, a system of optional paths is chosen "in advance" for every source–destination pair, and every packet for that pair must travel along one of these optional paths. Deterministic routing is thus a subset of oblivious routing since in deterministic routing, every source–destination pair has only one routing path. In what follows, we applied our algorithm to NoC with an oblivious routing scheme (named "OE-split" as described next) to show that the proposed buffer allocation algorithm also works under oblivious routing.

Similar to OE-fixed, OE-split is also developed based on minimum odd–even routing to ensure freedom from deadlock. In particular, if a packet with a given source and destination can be routed to both directions dir_1 and dir_2, it will have equal probability[8] to be routed to dir_1 and dir_2 in OE-split regardless of the current network condition. Since these optional routing paths for every source–destination pair and the probability of taking them can be precalculated, our algorithm can be applied to such systems without any modification.

The results of applying buffer allocation to 4 × 4 mesh NoCs using OE-split routing are shown in Table VIII. Similar to the results presented for the NoCs using the OE-fixed routing (Table VII), a significant performance improvement is achieved here through buffer allocation as well. This shows that our proposed buffer allocation approach can also be successfully applied to NoCs using oblivious routing.

Finally, we would like to point out that the proposed algorithm is also very fast. Take the buffer allocation for a 4 × 4 NoC with XY routing and $B = 192$ as an example (this corresponds to the data in the second and fourth columns in Table IV). The "run time" for generating the buffer allocation for each traffic pattern is less than 30 s, when running on a desktop with a Pentium-4 2.8-GHz processor.

C. Evaluations Under Realistic Traffic Conditions

We also evaluated the performance of our algorithm by applying it to applications that mimic real-world traffic. In particular, two realistic traffic scenarios were investigated. The first scenario (described in Section V-C1) includes a set of applications derived from "E3S" benchmark suites [11], as well as an audio–video benchmark by profiling a video/audio application. For the second scenario (described in Section V-C2), we use realistic bursty traffic pattern from a real video application to evaluate how the proposed buffer allocation algorithm performs under non-Poisson traffic pattern.

1) Evaluations Using E3S and Audio–Video Benchmark: Three benchmark applications are collected, namely "auto-indust," "telecom," and "audio–video." Both auto-indust and telecom are retrieved from E3S benchmark suites, 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 × 4 NoC. As an example, Fig. 11, shows the mapped communication task graph for our audio–video benchmark with the communication volume between different tasks derived through simulation using real video and audio clips as inputs. The communication rates between any two tiles are then 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. Moreover, since the proposed buffer size allocation makes all decisions locally (i.e., we only compare pairs of channels to determine which one has a higher blocking rate), the error caused by the approximations (in the modeling of the traffic and the queuing analysis approach itself) can be controlled.

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

[8]Note that since the OE-split serves only as an example to show that our algorithm can be applied to oblivious routing. Thus, the probability of taking different directions was determined arbitrarily (with equal probability for each possible direction). A more sophisticated oblivious routing can carefully determine the probability to minimize congestion.

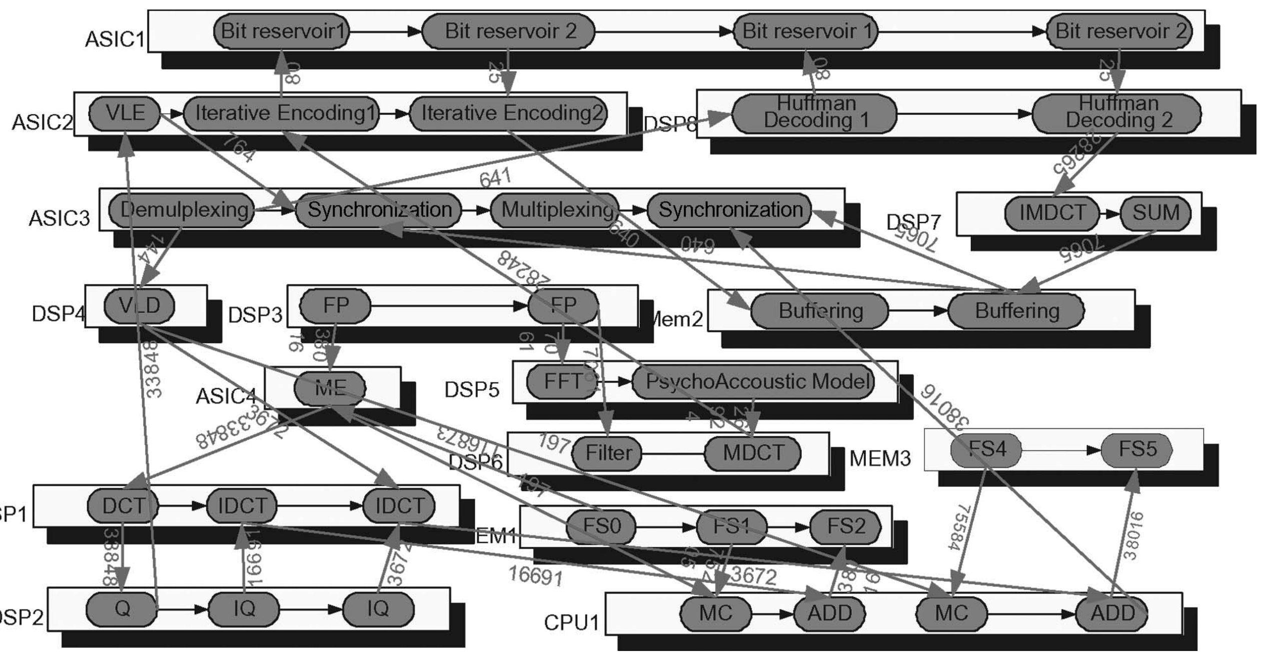


Fig. 11. Mapped communication task graph for audio–video benchmark on a 100-MHz NoC. Communication volumes shown in the figure have a unit of 10 kB, which represents the average communication volume between different tasks in 1 s.

TABLE IX
PACKET LATENCY (CYCLES) COMPARISON USING REALISTIC TRAFFIC PATTERNS (XY ROUTING, MESH TOPOLOGY, 4×4 NOC)

Benchmark	***UNoC*** **(B=96)**	***UNoC*** **(B=144)**	***CNoC*** **(B=96)**
auto-indust	1100.1	545.8	64.9
telecom	988.4	96.4	28.3
audio-video	1189.2	346.9	175.4

Comparing Table IX with Table II, it is clear that the buffer allocation is even more beneficial for these benchmarks. Indeed, compared to the random traffic patterns in Table II, 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, to achieve an equivalent or shorter packet latency as CNoC ($B = 96$), the average buffer length for a UNoC should be increased to 11, 11, and 10, respectively. This means that, to achieve the same performance as a NoC with customized buffer allocation, the NoC implementation with uniformly distributed buffer size will need to consume 4.5, 4.5, and 4 times more buffering spaces, respectively.

2) Evaluations Using Bursty Traffic: To show the benefits of the buffer allocation approach on real applications, we applied the proposed algorithm to a 4×4 NoC configuration that exhibits bursty traffic patterns generated by long traces extracted from simulating a real video application. On top of this bursty video traffic, we also added bursty traffic that represents another application, sporadic in nature, that may coexist with the main video application on the same NoC platform in practice. Indeed, while the video application is the one that runs continuously and represents the main challenge for the network, we may occasionally have other applications (e.g., encryption) that appear and disappear in the system at different timescales. Nevertheless, considering such occasional applications is important since they add an extra workload to the network and make the traffic pattern the most complex one can consider.

The proposed buffer allocation algorithm is used to customize the buffer sizes of the NoC. Both the UNoC with an average buffer size of $2 \times$ SP and the CNoC with the same amount of total buffering resources were then simulated using the same traffic trace. Simulation results show that UNoC has an average packet latency of 878.12 clock cycles, whereas CNoC has an average packet latency of only 76.79 clock cycles; this represents about one order of magnitude savings, which basically come from smarter buffer allocation alone. This result shows that the proposed buffer allocation algorithm is indeed beneficial for real-world applications characterized by non-Poisson traffic patterns.

D. Comparison With a Heuristic Algorithm

Since there is currently no other algorithm available for buffer size allocation in NoCs, we compared our algorithm with a simple heuristic algorithm. Given the total buffering space B, the heuristic algorithm simply allocates the buffer size of each input channel to be linearly proportional to the traffic load of that channel. Mathematically, $b_{x,y,\mathrm{dir}} \propto \lambda_{x,y,\mathrm{dir}}$. In what follows, we use "LPNoC" to denote the NoC whose buffer size is allocated using this heuristic algorithm ("LP" stands for linearly proportional).

Table X shows the average packet latency comparison between UNoC, CNoC, and LPNoC under random traffic patterns with a total buffering space budget $B = 192$. As we can see, LPNoC is able to achieve a better packet latency compared to UNoC with the same amount of buffering space, whereas CNoC

TABLE X
PACKET LATENCY (CYCLES) COMPARISON USING RANDOM TRAFFIC PATTERNS (XY ROUTING, MESH TOPOLOGY, 4×4 NOC)

Pattern	***UNoC*** **(B=192)**	***LPNoC*** **(B=192)**	***CNoC*** **(B=192)**
uniform	998.0	602.7	356.2
1hotspot-1	951.8	812.2	666.2
1hotspot-2	1026.9	218.9	36.7
2hotspot	1000.2	84.9	34.2
3hotspot	1053.9	385.0	116.4

always outperform LPNoC. This again shows the effectiveness of our proposed approach.

VI. LIMITATIONS AND EXTENSIONS

In this section, we summarize some basic assumptions we made in developing the approach presented in this paper. We also discuss the implications of such assumptions and possible extensions.

A. Switching Technique

In this paper, we assume that store-and-forward or virtual cut-through switching is used in the NoC. Thus, in our analysis, a packet is treated as a basic and atomic unit since it is always transmitted or buffered as an indivisible entity.

Wormhole switching is another popular switching technique [9], and it can be more suitable for implementing NoCs compared to store-and-forward and virtual cut-through switching. However, due to the inherent complexity involved in the performance analysis for NoCs with wormhole switching, it is not straightforward to directly extend the approach presented in this paper to address the buffer allocation problem for NoCs with wormhole switching. More precisely, for NoCs with wormhole switching, a new analytical model needs to be derived, as it is necessary to treat each flit (instead of a packet) as a separate "customer" in the queuing model. Moreover, the header flit and the payload flits can no longer be treated independently because the delivery of these flits are tightly coupled.

Nonetheless, we believe that the results presented in this paper can serve as a first step toward addressing the general buffer allocation problem. In addition, the approach developed in this paper also has its practical impact considering that some real-world NoC designs do use or support virtual cut-through switching (e.g., [3], [26], and [29]).

B. Traffic Model

Our approach is based on queuing theory with the assumption that the packets generated in each PE follow a Poisson distribution. Although this assumption may not hold for some applications, we believe that the use of the Poisson traffic model is still meaningful due to the following considerations:

1) While Poisson-based models are not necessarily accurate in modeling "all" applications, they can be used as a good approximation for quite a few classes of applications. In fact, the Poisson models are the "only" mathematical models where knowing just a single parameter (i.e., the lambda that characterizes the distribution) is enough to derive quite sophisticated performance models. This model parsimony made the Poisson models widely used in performance analysis, and there are tons of papers in very diverse application domains that are based on this stochastic assumption. Although more accurate models may be available for some particular applications, such models are usually very complicated and likely to behave poorly for some applications compared to Poisson models.
2) From a different perspective, although better modeling accuracy can be achieved with some sophisticated models for certain application domains, such models are usually very difficult to use in analyzing the system in a "global" manner. Taking the multimedia application as an example, traffic models based on theory of "self-similar" or "long-range dependent" stochastic process have been introduced to SoC design [33]. However, such a model can only be applied locally for the system performance analysis, whereas the model we propose in this paper can be used to analyze the system in a global fashion; this is critical for allocating the buffer sizes where all the buffering resources (not only a critical buffer) need to be correctly provisioned. In fact, a sophisticated analysis like the one presented in [33] can be successfully applied, *after* the Markovian analysis based on Poisson models has been completed. In other words, once we provision all the buffering resources based on this approximate (but global) model, we can refine the analysis by focusing only a certain buffer, which may be critical for the design at hand and use a more sophisticated model.
3) Last but not least, the class of non-Markovian models used in performance analysis (particularly, in buffer sizing) is much less explored and familiar compared to the standard Markovian models. For instance, even for video traffic, which is notoriously bursty, it is arguably true that one should always use other models (e.g., long-range dependence model) instead of Poisson models. For instance, in a recent book by Park and Willinger entitled *Self-Similar Network Traffic And Performance Analysis* (Wiley, 2000), there are authors who provide tangible theoretical and practical evidence showing that the Markovian models may be actually sufficient in modeling very sophisticated traffic effects if used in a smart manner (e.g., through hierarchical use and modulation). See Section 12.4 on pp. 306–316, for instance, in the above-mentioned book, for a detailed discussion.

Thus, without claiming that the model we use in this paper is bullet proof, we argue that the real issue is actually more related to the actual use of these models rather than the models *per se*. Given the tractability of the Poisson models, as well as the fact that this paper is only the starting point of future efforts in this direction, we believe that their use in this paper is quite justified.

C. Local Buffer Sizes

As discussed in Section III-A1, we assume the size of the local input buffer (i.e., the input channel that accepts incoming packets from the router's local PE) to be infinite. To achieve

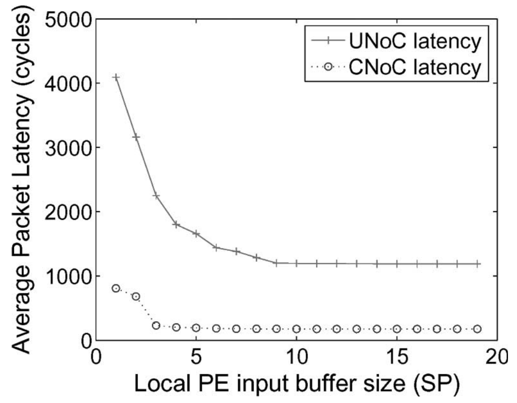


Fig. 12. Packet latency (cycles) comparison between UNoC and CNoC with finite local PE input buffers using an audio–video benchmark (XY routing, mesh topology, 4×4 NoC).

accurate results, the local buffer size at each PE needs to be taken into account. However, detailed PE models are highly application dependent and can vary significantly across different applications. On the other hand, reasonably large buffers at each PE may behave closely to a system with infinite buffering resources. Since for most applications the buffering resources at each PE are typically much larger than those available in the routers, it is reasonable to make the above assumption. Also, from a theoretical standpoint, this assumption makes this paper more interesting since, this way, we can derive an analytical solution for general NoC systems instead of providing mostly a design paper focused on a specific solution that is applicable to only one specific application.

Nevertheless, we show below how our buffer allocation algorithm performs on a NoC system with limited local input buffers using the audio–video benchmark presented in Section V-C1. In Fig. 12, we compare the performance of UNoCs and CNoCs under different local PE input buffer sizes. Similar to the experiment in Section V-C1, B is chosen to be 96 for both UNoC and CNoC such that each input channel in each channel has an average buffer size of two packets. As we can see, once the local PE input buffer size has reached a certain threshold (for this particular example, this threshold is about $9\times$ SP and $3 \times$ SP for UNoC and CNoC, respectively), increasing the local PE input buffer has no impact on the packet latency. Meanwhile, the packet latency of CNoC consistently outperforms that of UNoC with a significant margin, across different local PE input buffer sizes. This shows that our buffer allocation algorithm is indeed useful for NoCs with finite local PE input buffer sizes.

D. Extension for General Topologies and Other Routing Algorithms

Clearly, depending on the topologies and the routing algorithms an NoC system designer chooses, the traffic pattern across the network can vary significantly. Thus, the buffer allocation algorithm has to take into the consideration these two factors to achieve the best results.

In this paper, we applied our proposed approach to two topology examples: 2-D mesh and torus, as well as two different routing algorithms: namely XY and OE-fixed. It is important to note that our approach can be applied to arbitrary topologies, as well as any deterministic routing algorithms. The choice of the two topologies and routing algorithms is mainly justified by their popularity in many NoC design projects [10], [21], [23].

On the other hand, due to the interaction among topologies, routing algorithms, and buffer sizes of each channel, decisions on which topology to choose, which routing algorithm to adopt, and how to allocate the buffer size have to be made in parallel if the utmost optimality is needed. This, however, remains to be done as future work.

VII. Conclusion and Future Work

We have shown that, to minimize the implementation costs and maximize the NoC performance, the buffering size of each input channel has to be carefully allocated to match the application traffic characteristics. 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 a 2-D mesh network as the underlying NoC architecture serves mostly as an example. Indeed, our algorithm can be extended to arbitrary topologies by adapting the analytical models in Section IV-A accordingly to the target topology. Moreover, although we have evaluated our algorithm only for NoCs with XY, OE-fixed, and OE-split routing, the approach is general enough to be applied to other deterministic and oblivious routing schemes since, in these cases, the arrival rate of each channel can still be calculated as shown in (3).

We plan to advance this research in several directions. One possible direction is to extend this approach to NoCs that support adaptive routing. The main challenge comes from the difficulty involved in the calculation of the arrival rate for each channel as multiple routing paths are possible in adaptive routing. Another important extension is to accommodate NoCs with wormhole switching. Finally, we are working on a field-programmable gate array prototype and plan to use it for accurate evaluation of the effectiveness of the buffer allocation algorithm.

References

[1] V. S. Adve and M. K. Vernon, "Performance analysis of mesh interconnection networks with deterministic routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 3, pp. 225–246, Mar. 1994.

[2] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, no. 4, pp. 398–412, Oct. 1991.

[3] T. A. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Highly scalable network on chip for reconfigurable systems," in *Proc. Int. Symp. Syst.-on-Chip*, Nov. 2003, pp. 79–82.

[4] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

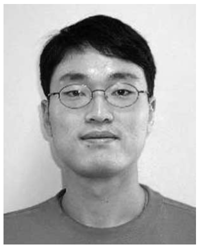
[5] J.-Y. L. Boudec and P. Thiran, *Network Calculus*, vol. 2050. New York: Springer-Verlag, 2001.

[6] V. Chandra, A. Xu, H. Schmit, and L. Pileggi, "An interconnect channel design for high performance integrated circuits," in *Proc. DATE Conf.*, Feb. 2004, pp. 1138–1143.

[7] G. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 729–738, Jul. 2000.

[8] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *Computer*, vol. 39, no. 6, pp. 775–785, Jun. 1990.

[9] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distrib. Comput.*, vol. 1, no. 3, pp. 187–196, 1986.
[10] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. DAC*, Jun. 2001, pp. 684–689.
[11] R. P. Dick, *Embedded System Synthesis Benchmarks Suites (e3s)*. [Online]. Available: http://www. ece.northwestern.edu/~dickrp/e3s/
[12] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. Int. Workshop Hardware/Software Codesign*, Mar. 1998, pp. 97–101.
[13] J. Dielissen, A. Radulescu, K. Goossens, and E. Rijpkema, "Concepts and implementation of the Philips network-on-chip," in *Proc. IP-Based SoC Design*, Nov. 2003.
[14] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *25 Years ISCA: Retrospectives and Reprint*, pp. 441–450, 1998.
[15] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on a chip: An architecture for billion transistor era," in *Proc. IEEE NorChip Conf.*, Nov. 2000, pp. 166–173.
[16] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 6th ed. New York: McGraw-Hill, 1995, ch. 15, pp. 661–732.
[17] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 4, pp. 551–562, Apr. 2005.
[18] ——, "DyAD—Smart routing for networks-on-chip," in *Proc. DAC*, Jun. 2004, pp. 260–263.
[19] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "xPipesCompiler: A tool for instantiating application-specific NoCs," in *Proc. DATE Conf.*, Feb. 2004, pp. 884–889.
[20] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Comput. Netw.*, vol. 3, no. 4, pp. 267–286, Sep. 1979.
[21] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J. Soininen, M. Forsell, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. Symp. VLSI*, Apr. 2002, pp. 105–112.
[22] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An architecture and compiler for scalable on-chip communication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 7, pp. 711–726, Jul. 2004.
[23] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A scalable, single-chip communications architecture," in *Proc. IEEE Int. Conf. Parallel Architectures and Compilation Tech.*, Oct. 2000, pp. 37–46.
[24] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Proc. DATE Conf.*, Feb. 2004, pp. 896–901.
[25] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, Feb. 1993.
[26] V. Nollet, T. Marescaux, and D. Verkest, "Operating-system controlled network on chip," in *Proc. DAC*, Jun. 2004, pp. 256–259.
[27] U. Y. Ogras and R. Marculescu, "'It's a small world after all': NoC performance optimization via long link insertion," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (Special Section on Hardware/Software Codesign and System Synthesis)*, vol. 14, no. 7, pp. 693–706, Jul. 2006.
[28] L. Peh and W. J. Dally, "A delay model for router micro-architectures," *IEEE Micro*, vol. 21, no. 1, pp. 26–34, Jan./Feb. 2001.
[29] E. Rijpkema, K. G. Gossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip," in *Proc. DATE Conf.*, Mar. 2003, pp. 294–302.
[30] I. Saastamoinen and J. N. M. Alho, "Buffer implementation for proteo networks-on-chip," in *Proc. Int. Symp. Circuits and Syst.*, May 2003, pp. 113–116.
[31] Semiconductor Association, *The International Technology Roadmap for Semicondutors (ITRS)*, 2004.
[32] L. G. Valiant, "A scheme for fast parallel communication," *SIAM J. Comput.*, vol. 11, no. 2, pp. 350–361, May 1982.
[33] G. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for mpeg-2 video applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 1, pp. 108–119, Jan. 2004.



Jingcao Hu (S'01–M'05) received the B.S. and M.S. degrees in electronics engineering from Tsinghua University, Beijing, China, in 1998 and 2000, respectively, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2005.

His current research focuses on logic synthesis and on-chip communication design methodologies.

Dr. Hu has received a Best Paper Award from the Design Automation and Test in Europe (DATE) Conference in 2003.

Umit Y. Ogras (S'00) received the B.S. degree in electrical engineering from Middle East Technical University, Ankara, Turkey, in 2000 and the M.S. degree in electrical engineering from Ohio State University, Columbus, in 2002. He is currently working toward the Ph.D. degree with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA.

His research focuses on communication-centric design methodologies for nanoscale systems-on-chip, with a special interest on networks-on-chip communication architectures.

Radu Marculescu (S'94–M'98) received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1998.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA. His current research focuses on developing design methodologies and software tools for system-on-a-chip design, on-chip communication, and ambient intelligence.

Dr. Marculescu is a member of the Association for Computing Machinery. He is a recipient of the National Science Foundation's CAREER Award (2001) in the area of design automation of electronic systems. He has received the 2005 Transactions on Very Large Scale Integration Systems (T-VLSI) Best Paper Award from the IEEE Circuits and Systems (CAS) Society, two Best Paper Awards from the Design Automation and Test in Europe (DATE) Conference in 2001 and 2003, and a Best Paper Award from Asia and South Pacific Design Automation Conference (ASP-DAC) in 2003. He was also awarded the Carnegie Institute of Technology's Ladd Research Award in 2002.