

# Cost-Performance Trade-offs in Networks on Chip: A Simulation-Based Approach

Santiago Gonzalez Pestana, Edwin Rijpkema, Andrei Rădulescu, Kees Goossens and Om Prakash Gangwal  
Philips Research Laboratories, Eindhoven, The Netherlands  
E-mail: [santiago.gonzalez.pestana@philips.com](mailto:santiago.gonzalez.pestana@philips.com)

## Abstract

*A challenge facing designers of systems on chip (SoC) containing networks on chip (NoC) is to find NoC instances that balance the cost (e.g. area) and performance (e.g. latency and throughput). In this paper we present a simulation-based approach to address this problem. We use XML to instantiate network components (routers, network interfaces) and their composition. NoCs are evaluated in terms of cost and performance by sweeping over different parameters (e.g. network topology, network interface queue depth). We then show, how we can obtain trade-off plots by using the results obtained with our simulation environment. Finally, by means of two examples we illustrate how trade-off plots can help the NoC designers in selecting the right network based on a set of different constraints.*

## 1 Introduction

The increasing complexity of systems-on-chip (SoC) together with the increasing wiring problems of newer IC technology generations make networks on chip (NoC) a promising replacement for busses and dedicated interconnect [3, 4, 9]. A NoC is an on-chip communication infrastructure that provides communication services to the IP blocks that connect to it. Examples of these services are connections with in-order data delivery and connections with end-to-end flow control.

NoCs are composed of *network interfaces* and *routers*. Network interfaces (NI) are the service access points of the network. These services are made available at one or more ports of the NIs. The routers provide the connectivity in the network; their job is to transport the data between the NIs.

The design of a NoC involves the design of NIs and routers together with the topology that specifies the way in which they are interconnected. The design space of designing NoCs is huge, spanned by the many parameters that describe the NIs, routers, and topology [13, 14]. Examples are the number of connections per NI, the depth of the queues in the routers, and the arity of a tree.

Chip design is typically characterized by tight cost constraints and high performance demands. Our goal is to help the designer in finding the right NoC. The right NoC is one that fulfills all cost and performance constraints and optimizes some other cost or performance metric. To fulfill the designer's need of making cost-performance trade-offs, we propose a method to obtain both cost and performance numbers of NoCs instances in the design space and a way to present these numbers.

We advocate an approach that relies on both VLSI study and simulation. The VLSI study is used to derive a model for the phys-

ical characteristics of a NoC, such as the silicon area and clock frequency of a NI as function of its parameters e.g. number of ports. Simulation then is used to generate performance numbers of the NoC as a whole. We use simulation for this because designing hardware for each NoC in the design space is unrealistic and, analytic methods make too many assumptions about the network and traffic to get accurate values for a real system.

To provide the performance numbers we have developed a simulator that can rapidly evaluate the performance of many NoC instances that are built from parameterized components. We automatically generate NoC instances for many points in the design space. These NoCs are specified in the XML and their performance is evaluated without the need to recompile the simulator.

To prove our approach, the paper describes a case study in which synthetic workloads are offered to the NoC. Synthetic workloads are used because it makes the system parameterizable. In the case study we evaluate the cost and performance numbers of a number of network instances and show how to use these numbers to find the best NoC that meets the user requirement.

The paper is organized as follows. Section 2 describes that we span the NoC design space in terms of parameterized network components and topology. Section 3 describes our approach to obtain the cost and performance numbers that allow the user to make design trade-offs. Section 4 deals with the metrics that are of interest to the NoC designer and describes a simple NoC cost model. Section 5 describes in detail the setup of a case study in which for a synthetic workload the best NoC must be found. Section 6 demonstrates our proposed approach: (1) cost and performance numbers are derived for a number of NoC instances, (2) the numbers are presented in a convenient way, and (3) the best network is selected for two example sets of requirements.

## 2 Network-on-chip design space

A network on chip (NoC) is composed of two types of components: routers (also called switches) and network interfaces (NI). Their composition is described by the topology in which these components are interconnected.

The NoC design space is spanned by choosing the router and NI architectures and the topology, and their corresponding parameters.

To illustrate the design space for a given router and NI architecture we use the network in Figure 1 as an example. The figure shows six IP blocks connected via four NIs to a four-router network having a mesh topology. In subsequent sections we describe the basic components and the topology to illustrate their parameterization.

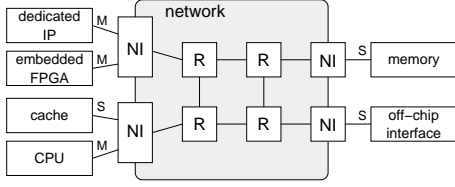


Figure 1. Example of SoC using a NoC.

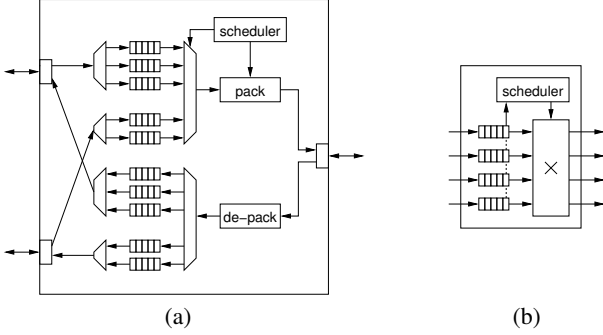


Figure 2. Example of NI architecture (a) and router architecture (b).

## 2.1 Network interface

A NI is described by two sets of parameters, i.e., parameters that describe the architecture of the NI and parameters that describe the instance of a given NI architecture. Two examples of the former set of parameters are whether services are connection based or not, and whether end-to-end flow-control is present or not. Clearly, what the set of the latter kind of parameters is, depends on what the actual architecture is. In this paper we focus on the latter set of parameters and fix the NI architecture to the  $\text{\AE}$ threal NI described in [13], of which Figure 2(a) shows an example.

This NI architecture provides connection-based services made accessible at one or more ports. At the ports, shown at the left-hand side of Figure 2(a), a transaction-level protocol is used. That is, IPs connected to these ports issue *read* or *write*-like transactions. A write transaction consists of the write command together with the data. The NI considers the (write-command,data) pair as a message to be sent over the network. For a read, the NI considers the read command and the corresponding response of the data as two separate messages. The size of a message is expressed in bytes. The relation between connections and messages is that related messages, e.g. an audio stream of (write-command,data) messages from an audio decoder to an off-chip interface, are written in the same connection queue. For this queue, properties like in-order lossless delivery are given, but also guaranteed throughput (not dealt with in this paper, see [12]) can be provided. The messages in the queues are scheduled and are subsequently packetized to be transferred over the network of routers. The NI has two queues per connection, one for outgoing and one for incoming traffic. The NoC services are accessible via one or more ports of the NI and there are one or more connections per port.

The parameters describing the instance of the NI architecture are: number of ports, number of connections for each of these ports individually, and depths of the queues for each of the connections individually. Moreover, the depths of the incoming and outgoing queues is parameterized individually as well. The NI instance in Figure 2(a) has two ports, three connections for the first

port and two connections for the second port. The depths of all the queues is set to five words, where a word is four bytes.

## 2.2 Router

A router has two sets of parameters similar to the NI. The set of parameters that describe the architecture include routing algorithm, routing mode, buffering architecture, and schedule algorithm. In this paper we fix the architecture to the  $\text{\AE}$ threal router architecture described in [11] of which an instance is shown in Figure 2(b). We only are interested in the Best Effort (BE) part of that router architecture.

The router is packet-switched and uses source routing, i.e. the routers use the route information that the NIs have put in the packets. It uses wormhole routing and input queuing.

The arity of the router is the number of input ports, which we take equal to the number of output ports. At every input port there is a queue that has a parameterized depth. So, parameters that describe the instance of the router architecture are arity and depth. The router instance in Figure 2(b) has arity four, and has queues of five words deep.

## 2.3 Network on chip

A NoC is composed of NI and router instances. To complete the NoC description we also must know the topology in which components are interconnected. In this paper we focus on regular topologies i.e., 2D-meshes and  $k$ -ary trees and  $k$ -ary fat-trees. Each of these topologies is parameterized by its own set of parameters. A 2D-mesh is parameterized in its width and height, and a tree and fat-tree by their arity and height [16].

In the network topologies envisaged in this paper we make the following assumptions. In all topologies the links are bidirectional. In meshes there is a NI at every router, and in  $k$ -ary (fat-)trees there are  $k$  NIs at every leaf router.

Note that the NoC design space as a whole is spanned by the selection of the topology and the NI and router architectures, and their respective parameters.

## 3 Networks on Chip design approach

In this section we propose a network-on-chip design-flow approach that guides the designer through the SoC design process allowing him to concentrate on the performance results. Performance results are obtained by using simulation results. Later in this section we address in detail the interface with the simulator in order to follow the proposed design approach. We use SystemC to [1] to build our simulator.

SystemC provides us different abstraction levels together with the flexibility to model, develop and simulate all parts composing a network on chip (e.g. routers, network interfaces) and external IP blocks connected to it. A number of simulators and design approach tuned to different applications (e.g. real time, shared-memory) can be found in the literature [7, 9, 15, 17]. A new simulator environment has been developed to provide a flexible compositional platform to experiment with a large number of components and their parameters. The environment natively is not oriented to any specific application or topology. Different applications using different services can be mapped without changing the system core. NoC parameters (defined by the user) define the NoC behavior. We use a transaction-level model (TLM), which allows us to describe hardware system components at a high level of abstraction. Although the results are less accurate than using register-transfer-level (RTL) simulations [10], we obtain higher simulation speed.

The simulator itself models the NoC at the level of flits, giving a good trade-off between simulation accuracy and speed.

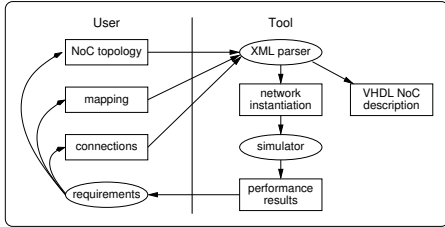


Figure 3. Design flow used in our environment.

To provide maximal flexibility to the designer in the parameter description, we propose the design flow shown in Figure 3. The designer concentrates on specifying three files (topology, mapping and connection) that are used by the XML parser to instantiate the SoC to be simulated. They describe the network and IP blocks along with their parameters. As an output we obtain a set of files containing performance results and statistics from the network. The designer can evaluate each NoC independently or automate the design flow by using our simulator to evaluate a large number of topologies, analyze them and select the most appropriate for a given set of constraints.

If the constraints are not (satisfactory) met (e.g. cost-performance requirements) the NoC designer can steer design decisions (e.g. mapping). In the next section we explain how we instantiate a general SoC design with the three files, which contain the parameters of all components and how they are interconnected in the NoC.

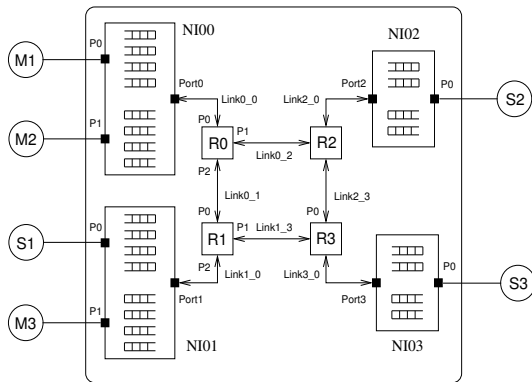


Figure 4. Example of SoC.

### 3.1 Interface to the simulator

The interface to the simulator environment is based on three files. The files are described using the Extensible Markup Language (XML). XML has among other features the advantages of being a standard language, it is suitable to organize and structure data, it is human readable, and parsers are available to automate the instantiation process. The XML system-on-chip description provided by the user is also used to generate VHDL code to have consistent views of the simulator and hardware implementation. The main motivation for using an XML-based description of NoC is twofold (a) flexibility to describe NoC components and (b) avoiding long re-compilation times, because we instantiate components at run time.

The use of three files to describe every component and their interrelations allow us to come with a flexible and well-defined

way to decouple network infrastructure components (e.g. routers, network interfaces) from user-defined modules (e.g. traffic generators, IP blocks). A set of basic components to experiment with (e.g. routers, network interfaces and traffic generators) are provided. Additionally, the user can supply a library of IP blocks or network elements that can be configured by means of attributes in the XML file and be instantiated at run time. The description and content of the following files use Figure 4 as an example.

**Topology file:** It contains the NoC topology description. We specify the routers and network interfaces, and the way they are interconnected (eg. mesh, ring, fat tree). The following parameters are given in the description. For the router, the type of router (e.g. input queuing, virtual output queuing), queue depth, number of input/output ports. For the network interface we specify the number of network interface ports, the number of connections and the router to which the network interface is connected.

In Figure 5 we can see part of the topology definition for the system shown in Figure 4. We have defined router “R0”, which is a 3-arity input-queued router, with a queue depth of 12 words. From the description see that “Link0\_0” connects network interface “NI00” to “R0” (links are bidirectional). A network interface can connect more than one IP block. In our example “NI00” contains one master<sup>1</sup> IP block as well as one slave IP block.

```
<RouterIQ id="R0" iq="12">
  <R_Port id="P0" link="Link0_0"/>
  <R_Port id="P1" link="Link0_2"/>
  <R_Port id="P2" link="Link0_1"/>
</RouterIQ>
<NI id="NI00">
  <NI_Port id="Port0" link="Link0_0"/>
  <MasterNIP id="P0" connections="2"/>
  <SlaveNIP id="P1" connections="2"/>
</NI>
<NI id="NI02">
  <NI_Port id="Port2" link="Link2_0"/>
  <SlaveNIP id="P0" connections="2"/>
</NI>
```

Figure 5. Example of a topology description.

**Mapping file:** Once the topology has been selected, we have to describe how IP blocks are going to be connected to the NoC. The mapping file provides a description of which blocks are connected to which network interface. In the same way as in the topology file, XML attributes are used to describe parameters that can be modified easily without the need of changing the simulator code.

Part of the XML mapping file is shown in Figure 6. We have declared a master module of type “TG” (Traffic Generator) with some extra parameters such as the type of traffic distribution (normal distribution), speed, or message size. This master module is connected to the NI “NI00” via the port “P0”.

```
<Master id="M1" type="TG" DIST="Nor" SPEED="8" SZ="128">
  <MasterNIP ni="NI00" id="P0"/>
</Master>
<Slave id="S3" type="Memory">
  <SlaveNIP ni="NI03" id="P0"/>
</Slave>
```

Figure 6. Mapping description example.

**Connection File:** To characterize the system completely, a set of connections is specified. Connections are established between one master network interface port and one or more slave network

<sup>1</sup>A master starts a transaction which is executed by the slave.

interface ports. Currently static connections are used, but in the future, our simulation environment will allow dynamic handling of connections. We use connections to connect one master IP block to one slave IP block; although other combinations are possible [12] we only use the most simple of them. Connections are opened with a set of properties (e.g. message integrity, transaction completion, transaction ordering) and parameters (e.g. routing path). The simulator provides a shortest path routing algorithm in case no path is specified in the file.

Figure 7 shows several connection configurations. Masters and slaves have their own connections identifiers for a given connection. As an example we show a connection between M2 and S3, the master identifies that connection with id 3 and the slave with id 2. The channel properties are Guaranteed Throughput (GT) for outgoing traffic and Best Effort (BE) for incoming traffic. For GT traffic we have to specify the slots we reserve to obtain the required bandwidth/latency guarantees (e.g slots="2 5"). Although we have flexibility to use GT traffic, in our experiments we only consider BE traffic.

---

```
<Connection master="M2" cidm="3" slave="S3" cids="2">
  <Request type="GT" slots="2 5" path="1 0"/>
  <Response type="BE" path="1 0"/>
</Connection>
<Connection master="M3" cidm="2" slave="S2" cids="3">
</Connection>
```

---

Figure 7. Part of the description of a connection file.

## 4 Cost and performance metrics

Due to the large design space of NoCs, obtaining a good balance of parameters such as cost and performance is a major challenge [2, 5, 8]. To find this balance the system designer needs to trade off cost and performance numbers to find, for example, the lowest-cost NoC that meets all performance requirements or the NoC that provides the highest throughput but still meets latency and area constraints.

What metrics should be optimized normally depend on the application that makes use of the NoC. For example, in portable devices the main concern typically is area and/or power, whereas for high-end video processing throughput and latency are more important.

In previous section we described how to obtain performance numbers, but we also must answer the question what exactly has to be traded off. This means that we need to clearly define the cost and performance metrics. The cost and performance numbers depend on the selected network components and the topology, and the setting of their corresponding parameters. Section 4.1 deals with cost metrics and introduces a simple cost model. Section 4.2 deals with performance metrics.

### 4.1 Cost metrics and cost model

NoC cost metrics relate to the physical aspects of the NoC. Examples are silicon area, power consumption, and bit-error rate due to cross-talk and external influences. As cost modeling is work in progress, we use for this paper a simple cost model for the silicon area metric. Silicon area is measured in  $mm^2$ .

To model the area of the NoC we use a zero wire cost model, that is, the area of the NoC is taken to be equal to total area of its components. The wires area caused by interconnecting the components are thus ignored in this paper.

In Section 2 we have seen that the network components, i.e. the routers and NIs, are parameterized. To allow a rapid evaluation of

the NoC area, we propose to derive the area of the router and NI as function of their parameters. For a particular NoC instance all parameters are fixed and these values then can be used in the area functions of the individual components.

The area functions are obtained by synthesis of RTL models of the router and NI architecture. We have done this for the architectures introduced in Section 2 as follows. We have a basic assumption on the form of the cost function as function of the parameters of interest. For example consider the router area as function of its arity. There are parts of the router whose area is linear in the arity (e.g. queuing) and parts that are quadratic in the arity (e.g. the switch inside the router), e.g. there is a part of the router that is linear in the arity (the control and queuing path) and a part that is quadratic in the arity (the switch). So, we expect the area to be of the form  $\alpha_1 \cdot a^2 + \alpha_2 \cdot a + \alpha_3$ . By synthesizing for several values of the arity we compute the best values for  $\alpha_1, \alpha_2$  and  $\alpha_3$ .

A router having 48 words input queue depth has been synthesized in a  $0.13\mu m$  CMOS process for a clock frequency of 500 MHz and data path width of 32 bits (for each link/port). The resulting cost model given in terms of arity  $a$  is:

$$A_R(a) = 0.808a^2 + 23a \quad (10^{-3}mm^2) \quad (1)$$

Similarly, let  $p$  be the number of ports of the NI,  $c$  be the number of connections per port, and  $q$  be the depths of the queues in the NI. By synthesizing for several values of number of ports, connections per port, and queue depths  $q$  (with a frequency of 500 MHz and 32 bits wide data path) we derived the following area function.

$$A_{NI}(p, c, q) = 19.6pc + 0.72pcq + 4.8 \quad (10^{-3}mm^2) \quad (2)$$

### 4.2 Performance Metrics

There are several performance metrics that may be of importance. For the system designer that must select a NoC these are the metrics that reflect the total system behavior at the level of the offered services. Those are, throughput offered at the NI ports and end-to-end message latency.

**Throughput** is measured at the ports of the network. It is the average amount of user data that is accepted by the network on that port in a certain amount of time. Aggregate throughput is the sum of the throughputs at all network ports. Throughput is measured in bytes per second.

**Latency** is defined as the difference between the time at which the first word of a message has been offered to the network and the time the last word has been delivered to the destination. That is the so-called *total latency* ( $L_T$ ). Total latency is composed of the **waiting latency** ( $L_w$ ) and **network latency** ( $L_{net}$ ). If the NI has no space to accept data coming from the IP, the data is queued outside the network. The **offered load** (data produced by the IP) may be higher than the **accepted load** (i.e. accepted by the NI), and  $L_w \geq 0$ .

## 5 Case study: set-up

In this and the next section we use a case study to demonstrate our approach. The goal is to find the best network that meets the user requirements for a given application. We first describe the application model that we use. Then we describe the portion of the design space that we want to explore. Finally the cost and performance numbers are presented and their usefulness in selecting the right network is demonstrated.

## 5.1 Application model

In this case study the focus is on data streaming. To allow a simple scaling of the application, we have chosen a synthetic workload. This workload is generated by *traffic generators*. A traffic generator is the combination of a *traffic master* and a *traffic slave*.

In this paper we fix the write transactions data payload to 8 bytes. When these transaction are issued and to whom is described by the *temporal* and *spatial distribution*, respectively. In order to let the network not influence these distributions a master has a (logically) infinite buffer for each slave it communicates with. We also assume that slaves are infinite fast to avoid IP-NoC interactions (e.g. shortage of flow-control credits).

The temporal distribution describes the time at which new write transactions are issued. We use a normal distribution  $N(\mu, \sigma)$ , where  $\mu$  is the mean inter-arrival rate and  $\sigma$  is the variance. The mean  $\mu$  is directly derived from the required throughput, e.g., to let the traffic master produce data at a rate of 200 Mbyte/s, the value of  $\mu$  is  $200/8 = 25$  mega transaction per second. We further fix the variance  $\sigma$  to 10% to create some jitter in the data production rate.

The spatial distribution of a traffic master describes how the transactions are distributed toward the slaves. We use a uniform distribution.

To connect a single traffic generator to a network interface we require two network interface ports. Each traffic master issues write transactions to the traffic slave of every other traffic generator. The masters communicate to the slaves via connections. There is thus a single connection for each master-slave pair.

## 5.2 Experiments to realize

In this paper there are 16 traffic generators that communicate as just described. Note that we thus require  $16 \cdot 15 = 240$  connections which is more than what is found for typical applications of this size. Since each traffic generator requires two network interface ports, we thus must find the best network that has 32 network interface ports.

To find this network we will vary three free NoC parameters, i.e., network topology, number of ports per NI, and depth of the buffers in the NIs. Thus, a NoC is completely characterized by the triple  $\langle \text{topology}, \# \text{NIPs}, \text{queue depth} \rangle$  and we refer to them using this notation. E.g., the mesh network instantiated with 4 ports per NI and queue depth of 8 is referred to as  $\langle \text{mesh}, 4, 8 \rangle$ .

The topologies we evaluate are meshes, binary trees, and fat-trees. Because the number of total ports is constant (32), the size of these topologies depends on the number of ports per network interface. The number of ports per network interface does not significantly change the total NI cost but it does influence the size of the network. We evaluate cost and performance for networks with 2 and 4 ports per network interface.

The buffer depth in the NIs plays a paramount role both in cost and performance. We evaluate cost and performance with NI queues with a depth of 4 and 8 words.

For a mesh with four ports per NI we take a  $3 \times 3$ -mesh and do not connect an NI to one of the routers at the corner of the mesh, this gives us a total of 8 NIs and thus 32 NI ports in total (assuming one NI per router). With two ports per NI we take a  $4 \times 4$ -mesh, having one NI per router, and 2 ports per NI thus in total we have 32 NI ports in total.

For the binary (fat-)tree with four ports per NI we need three levels of routers. With two ports per NI, we need four levels. For

both topologies we assume there are two NIs per router at the bottom of the tree.

## 6 Case study: results

In this section we illustrate with an example the design approach and methodology that we proposed in this paper. We use our cost model and simulation environment to rapidly generate cost and performance numbers for the cases showed in previous section. To allow the selection of the right network the total set of cost and performance numbers are presented in a single scatter plot.

To understand the trade-offs that are captured in the scatter plot we first present performance numbers of a single topology for different number of NI ports and different NI queue depths. Moreover, these performance numbers also explain how the scatter plot is obtained.

We present NoC performance in the so-called Chaos Normal Form (CNF) [6]. In CNF performance is presented by pair of plots: one being the *total accepted load* versus *total offered load* plot and the other being the *latency* versus *total offered load*. Total offered load is the sum of the loads offered at every network interface port, and total accepted load is the part of the offered load that the NIs actually accept.

Figures 8 and 9 show the performance numbers of the  $\langle \text{mesh}, p, d \rangle$  for  $p = 2, 4$  and  $d = 4, 8$ . In Figure 8, all curves start linearly with accepted load equal to offered load. The point at which the curves start to bend is the saturation point of the NoCs. We observe that both fewer ports per NI and deeper queues in the NI results in a higher saturation point and hence total accepted load.

In Figure 9 we can observe a steep increase in latency close to the saturation point. Typically networks are used below the saturation point to obtain acceptable latency. The pair of CNF plots can be used to obtain the maximum throughput for a given latency constraint. Figure 9 is used to find the offered load given the latency. The accepted load for this latency is equal to the offered load when we are below the saturation point.

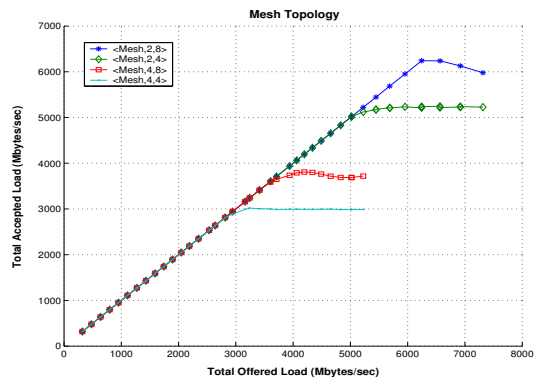


Figure 8. Total accepted load versus Total offered load for different mesh instances.

By using Figures 8 and 9 for our three topologies of interest we have derived a scatter plot as it is shown in Figure 10. For various latency bounds we use the NoC performance in CNF to obtain the accepted load. For each network we use the cost model to compute the NoC area. As each different NoC has its own area, the horizontal axis plots both area ( $mm^2$ ) and NoC instance. We

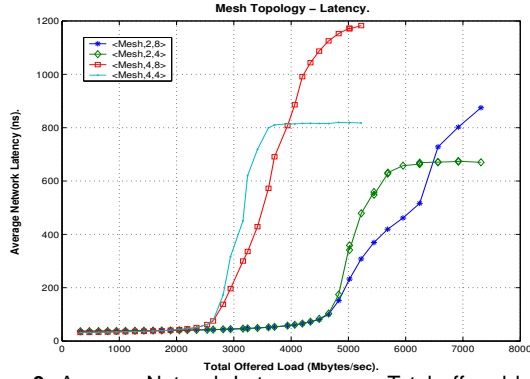


Figure 9. Average Network Latency versus Total offered load for different mesh instances.

have joined all points corresponding to the same latency constraint by a line for readability.

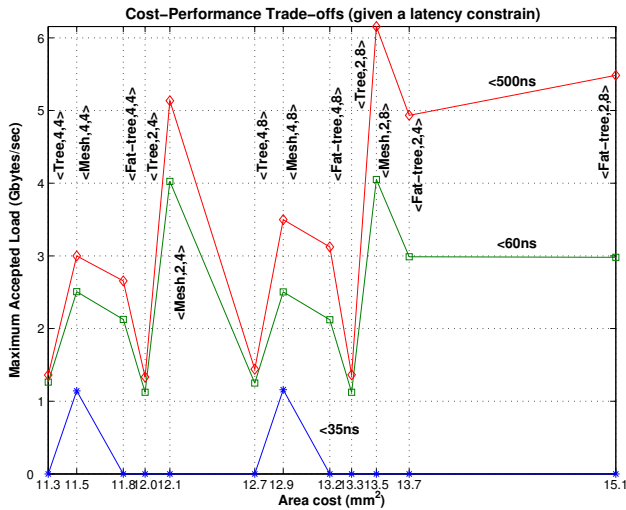


Figure 10. Cost performance trade-off.

Due to the all-to-all communication pattern of the case study, the total number of buffers is quadratical in the total number of NI ports, hence the cost figures plotted in Figure 10 are unrealistically high but they can be used to illustrate the method. In typical applications we expect an almost linear dependence, and therefore lower cost.

The user can achieve cost-performance trade-offs in several ways. Typically, a system designer must meet a set of constraints and wants to optimize for some other parameters. Below we give two examples and use the scatter plot in Figure 10 to find the best network.

**Example 1:** Find the lowest area network with the constraints that it provides a total accepted load of at least 2Gbyte/s with an average latency of at most 500ns. From the 500ns curve we select the left most point above the load=2Gbyte/s line, i.e. <mesh,4,4>. Note that because our simulation environment collects data as function of offered load, we can extract more latency bounds that the ones represented in Figure 10, without the need of re-simulation.

**Example 2:** Find the NoC that provides the highest accepted load, constrained by an area of  $12.5\text{mm}^2$  and an average latency of 60ns. From the 60ns curve we select the highest accepted load at the left of the area= $12.5\text{mm}^2$  line, i.e. <mesh,2,4> giving an

accepted load of 4Gbyte/s.

## 7 Conclusions

In this paper, we propose a method to help the designer in finding the right NoC given a set of constraints. To do so, we first introduce cost and performance metrics that allow a quantitative evaluation of NoCs. We focus on the metrics that are visible to the NoC user, such as silicon area, latency, and offered/accepted load.

To obtain both cost and performance numbers we use VLSI study to provide NI and router areas as functions of arity and buffer sizes, and number of ports and connections and buffer sizes, respectively. NoC cost is obtained by summing (assuming zero-cost wires) the cost of the NIs and routers. Performance numbers (latency and throughput) are obtained using simulations. The simulator is optimized for fast simulation of many NoC instances; it uses run-time network instantiation, and both transaction-level modeling and flit-level simulation.

The performance numbers obtained from the simulation and cost numbers from the cost model are used to demonstrate the trade-offs of performance (throughput and latency) versus area. The cost-performance numbers allow the trade-offs for various requirements. We demonstrate this with a case study in which the best network must be selected for a synthetic workload.

## References

- [1] *Functional Specification for SystemC 2.0*. www.systemc.org, 2001.
- [2] G. Apostolopoulos et al. Quality of service based routing: a performance perspective. In *SIGCOMM '98*, pages 17–28, 1998.
- [3] L. Benini et al. Powering networks on chips. *ISSS*, pages 33–38, 2001.
- [4] L. Benini et al. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.
- [5] E. Bolotin et al. QNoC: QoS architecture and design process for network on chip. *The Journal of Systems Architecture*, Dec. 2003.
- [6] J. Duato et al. *Interconnection Networks, An Engineering Approach*. Morgan Kaufmann, 2003.
- [7] M. Forsell. Advanced simulation environment for shared memory network-on-chips. In *20th IEEE Norchip Conference*, 2002.
- [8] J. Hu et al. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *ASP-DAC 2003*, 2003.
- [9] S. Kumar et al. A network on chip architecture and design methodology. In *ISVLSI*, 2002.
- [10] S. Pasricha. Transaction level modelling of SoC with SystemC 2.0. 2003.
- [11] E. Rijpkema et al. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. *DATE*, 2003.
- [12] A. Rădulescu et al. Communication services for networks on chip. In S. Bhattacharyya, E. Deprettere, and J. Teich, editors, *Domain-Specific Embedded Multiprocessors*. Marcel Dekker, 2004.
- [13] A. Rădulescu et al. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. In *DATE*, 2004.
- [14] I. Saastamoinen et al. Buffer implementation for Proteo network-on-chip. In *ISCAS*, pages 113–116, vol.2, 2003.
- [15] G. Varatkar et al. Traffic analysis for on-chip networks design of multimedia applications. In *DAC*, 2002.
- [16] A. Varma and C. Raghavendra. *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*. 1994.
- [17] D. Wiklund et al. SoCBUS: Switched network on chip for hard real time embedded systems. In *IPDPS*, 2003.