Models of Computation for Networks on Chip

Axel Jantsch
Royal Institute of Technology, Stockholm, Sweden

Abstract

Networks on chip platforms offer the opportunity to introduce a new abstraction level that defines a set of platform services with performance and power characteristics. By making the implementation of these services entirely irrelevant for system design, an effective separation of system design from component design can be achieved.

We discuss the principles to formulate network-on-chip services to establish an abstract computational model that exposes all relevant properties of the platform's functionality, performance and power consumption while hiding all irrelevant implementation details. As in many other successful abstractions, these principles are based on separating functionality from time and power aspects to allow for reasoning about these properties at the system level.

As a concrete example we formulate a MoC for the Nostrum NoC. It is based on guaranteed bandwidth (GB) and best effort (BE) traffic. The MoC characterizes both GB and BE traffic in terms of closed formulas and allows for efficient composition of traffic.

1. Introduction

With the growing number of hardware and software intellectual property (IP) blocks on a single chip, system analysis, verification and integration has emerged as a main challenge due to the enormous amount of details that have to be handled. Abstraction can dramatically reduce the detailed data and allow for focusing on important properties of the system. However, an abstraction is not as good as any other and good abstractions are hard to devise. A good abstraction efficiently eliminates all irrelevant details, is still sufficiently accurate and gives rise to effective design and verification techniques.

The search for better design methodologies, methods and tools has always also been a quest for new and better abstraction levels to represent designs. The phenomenal growth of complexity has necessitated new and efficient ways to represent design aspects and design properties of concern while suppressing irrelevant but confusing details. Hierarchy and abstraction have been the two main tools to expose to the designer a limited view of a given design. In contrast to hierarchy, which selectively exhibits or hides design details, abstraction provides a new semantic model that is distinct from the lower level model but should be compatible with it and significantly simpler. Gates, register transfers, buses, instruction sets, procedures and objects are all successful examples of abstractions in hardware and software design that provide to tool developers and designers a semantically well defined layer, which hides lower level implementation details and which allows for building complex systems on top of it.

This quest continues and on today's agenda is the development of new and efficient abstractions for complex, multi core systems on chip (SoC) with diverse hardware components, sophisticated services and a layered and irregular software architecture. Figures 1- 4 show examples of complex state of the art SoCs covering various applications such as multi media, network processors and emulators. They all exhibit significant complexity and consist of various and diverse HW and SW components. The number of HW blocks range from around ten to several hundred and it is predicted to grow steadily over the next ten years. All these examples constitute sophisticated platforms that can be configured and programmed by application designers. However, this is a formidable task since the platforms offer not only a huge number of possibilities to map and implement a given functionality on the numerous and heterogeneous resources, they also offer software controlled caches, scratchpads and other memory structures, a variety of communication features, dynamic voltage and frequency scaling of components, and other features to tune delay, throughput and power consumption.

These kind of platforms need to be represented at a high level for two kind of users. (a) Platform developers design and extend a platform with a specific application domain in mind. In order to find a good architecture that provides the required performance at minimum cost and power consumption, the platform functionality and key non-functional figures of merit such as delay, bandwidth, power consumption have to be modeled and analyzed thoroughly. (b) Application developers map a given application onto the plat-

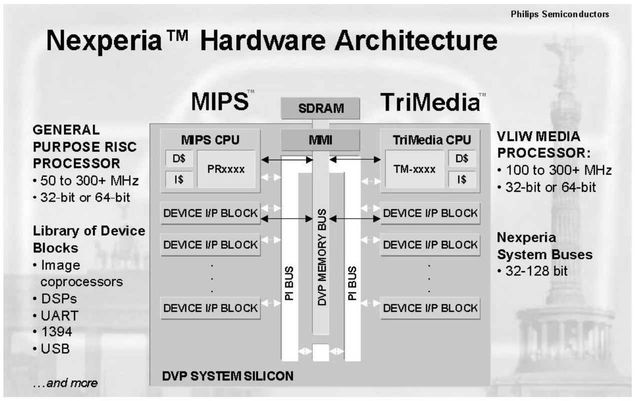

Figure 1. The Nexperia Platform from Philips Semiconductors.

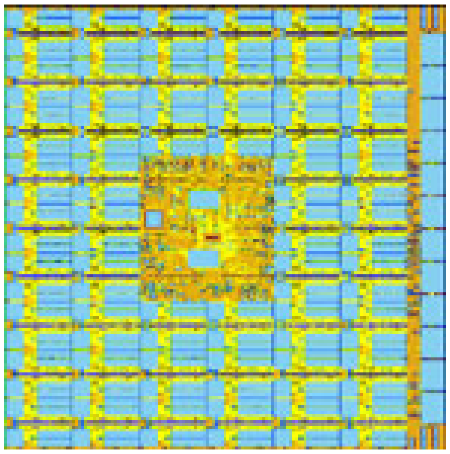

Figure 2. An IBM emulator ASIC with 768 processors for the Palladium II emulation system from Cadence.

form. Again, to validate the functionality and all relevant performance and cost figures of the complete system the platform together with the application has to be modeled in a way that exposes the right information. Note, that in both situations the detailed implementation of the platform, e.g. which technology and which memory components, are probably not known. The models and the analysis should be valid for an entire class of possible implementations. What is known is an abstract definition of the platform.

An abstraction is not as good as any other and the main question is which properties of the implementations should be exposed to system designers. These properties should be

- *abstract* in that many different implementations shall be able to provide them;
- *relevant and meaningful* to the system designer to facilitate the systematic exploration of the design space;
- *stable and reliable* even if significant functional and implementation changes are introduced later. We will elaborate on this in section 3 under the term composability.

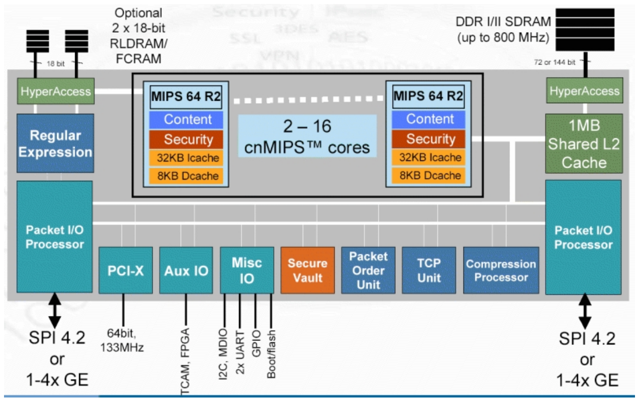

Figure 3. The Octeon network processor from Caveon with up to 16 processor cores.

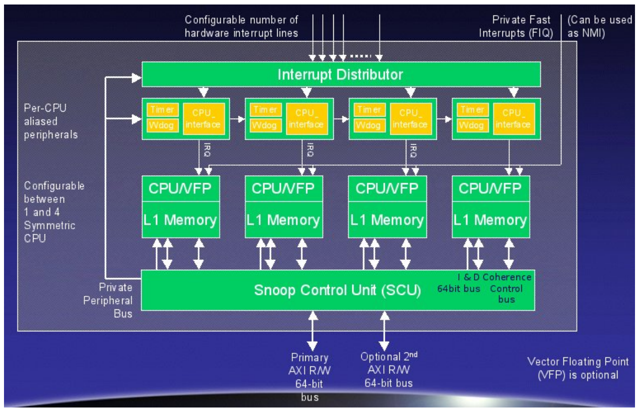
Configurable number of hardware interrupt lines
Private Fast Interrupts (FIQ)
(Can be used as NMI)
Interrupt Distributor
Per-CPU aliased peripherals
Timer
Wdog
CPU interface
IRQ
Configurable between 1 and 4 Symmetric CPU
CPU/VFP
L1 Memory
Private Peripheral Bus
Snoop Control Unit (SCU)
64bit bus
I & D Coherence Control bus
Primary AXI R/W 64-bit bus
Optional 2nd AXI R/W 64-bit bus
Vector Floating Point (VFP) is optional

Figure 4. A symmetric multi-processor from ARM.

We suggest that the key concept behind the Models of Computation (MOC) point to a promising direction. To substantiate this idea section 2 elaborates the main concepts and the history of models of computation. Section 3 introduces the concept of composability, that is a key property for any durable system level abstraction. Then we suggest that Network on Chip (NoC) is a useful concept to capture the essence of future highly complex, heterogeneous multi-core SOC platforms. We introduce Nostrum as a specific example of a Network on Chip, to serve as a vehicle for demonstrating how a MoC can be formulated for our purpose.

2. Models of Computation

A Model of Computation (MoC) is an abstraction of a computing device. From the early days of computing researchers have devised various MoCs for different purposes. One of the early questions raised in the 1920s and 1930s was "What is computable?" The Turing machine and the lambda calculus are two prominent examples of MoCs that were devised to answer this question [16]. They are wonderful examples of what a MoC can be in that they are abstractions of all conceivable implementations of a computing device entirely independent of the implementation technology or architecture, which may have a mechanical, an electronic or a biological basis. At the same time they represent relevant properties of all these possible computing devices in a concise and analyzable form. These MoCs have successfully answered above question and defined what is computable by any realization of a computing device.[1]

Later on researchers paid more attention to the efficiency of computations and asked, "How much time and space does a computation require?" The theory of algorithmic complexity answers this question in a very general and abstract manner. Based on an abstract notion of an algorithm, which is again independent of any specific implementation, different complexity classes of algorithms are identified. Each class is characterized by the type of function that describes the run time of the algorithm on any computing device. The runtime of an algorithm with polynomial complexity grows according to a polynomial function with the size of the input. Complexity theory states that an algorithm with polynomial complexity requires at least polynomial runtime on any sequential computing device. Other complexity classes characterize algorithms with constant runtime or exponential runtime. The MoC employed by complexity theory is simple and abstract and assigns an abstract time unit to each basic operation of an algorithm. The power of this idea stems from the fact that the performance of an algorithm can be described entirely independent of the implementation. Indeed, the performance figure, i.e. the algorithmic complexity, is valid for all conceivable implementations. Moreover, not only algorithms can be classified into complexity classes but even problems. Complexity theory states that, for a given problem in the complexity class with exponential runtime there exists no algorithm that can solve this problem in polynomial time [7].

An implication of the generality of this theory is that any computing device can simulate any other with at most a polynomial overhead in runtime [18]. Parallel architectures are included in this theory by relating time to space resources. This relation is captured by the parallel computation thesis that states that whatever can be solved in polynomially bounded space on a reasonable sequential machine model, can be solved in polynomially bounded time on a reasonable parallel machine, and vice versa [18].

[1]The term "Model of Computation" has only be used in the 1970s but the main ideas and concepts can certainly be traced back to the 1920s.

While complexity theory is powerful and offers general results, it is not very accurate. It cannot tell us how long, in terms of seconds, a given algorithm will run on a given computer. To be more accurate we need more specific MoCs. For sequential computers the *Random Access Machine* (*RAM*) [4] has been for a long time a general and fairly accurate MoC that has allowed to analyze performance properties of algorithms in an implementation independent way. It has captured well all interesting nonfunctional properties of most sequential computers. Over the years however, processor architectures and memory hierarchies have accumulated more sophisticated features and have consequently deviated more and more from the ideal RAM model. Thus, today the RAM model is not very helpful in analyzing the performance of a modern processor.

The parallel computer community has from the start faced the same challenges. The dependence of system level performance figures on architectural details of the parallel computer has prohibited the formulation of a MoC that is architecture and implementation independent and still allows to accurately analyze system performance [11]. The most widely used model is the *Parallel Random Access Machine* (*PRAM*) [6]. A number of processors execute in a lock-step way, that is, synchronized after each cycle governed by a global clock, and access global, shared memory simultaneously within one cycle. The PRAM model's main virtue is its simplicity but it poorly captures the costs associated with computing. Thus, the developer of parallel algorithms does not have sufficient information from the PRAM model alone to develop efficient algorithms. He or she has to consult the specific cost models of the target machine.

Many PRAM variants have been developed to more realistically reflect real cost [6, 2, 8, 17, 1, 3]. Some modeled the memory hierarchy and the memory access time more accurately while others reflect the communication and synchronization costs better. However, from all these attempts we can clearly observe the trade off between generality and accuracy. General models are not very accurate and accurate models are very specific and narrow in their scope.

For more detailed and elaborate treatment of the subject the reader is referred to [11, 9, 16, 15, 20], but from this very rough survey of MoCs we can observe the following points.

- As computing devices become more complex, diverse and heterogeneous the discord between generality and accuracy deepens, which renders very general models less and less useful.

- As MoCs get more specialized they have to be tailored for a specific architectural platform *and* for a specific purpose. It is unlikely that the same abstract MoC will be equally helpful in analyzing delay, power consumption, schedulability and functional correctness.

- The potential utility of a well designed MoC is substantial. The examples given above and many others demonstrate that MoCs can give significant insight into a *class of architectures and problems* and can form a sold basis for valuable techniques and tools.

Before we become practical by formulating a concrete MoC for a specific platform we highlight a crucial property that deserves special attention.

3. Composability

When we say a design paltform is *scalable* we imply that essential properties such as system performance and communication delay are not directly affected by the size of the design. For instance if the communication delay between a processor and memory slows down when we add new peripheral components, new memory or new processors, we conclude that the scalability is limited and designs beyond a certain size are infeasible. To capture this idea more precises the *arbitrary composability property* has been introduced [10] as follows.

> **Arbitrary composability property:** *Given is a set of components and a set of combinators which allow to connect and integrate the components into larger component assemblages. Components and combinators together are arbitrarily composable if, when a given component assemblage* **A** *can be extended with a component by using a combinator, the relevant behavior of* **A** *is never changed.*

Please note, that this is an engineering heuristics, not a formal mathematical property. As such it is and ideal and can be achieved to a higher or lower degree.

Note further, that this property is defined with respect to a *relevant behavior*. Thus depending on the given objectives and definition of behavior, the same components and combinators may or may not have the arbitrary composability property.

A prominent example of an arbitrarily composable system is a standard library of logic gates accompanied by appropriate design and layout tools. Adding a new NAND gate to an existing netlist will not change the behavior of the existing netlist. Merging two netlists is fine because the behavior of the two original netlists will not be affected. On the other hand a set of IP cores connected to a bus does not have this property. Adding a new core to the bus probably has an impact on the communication among the other cores

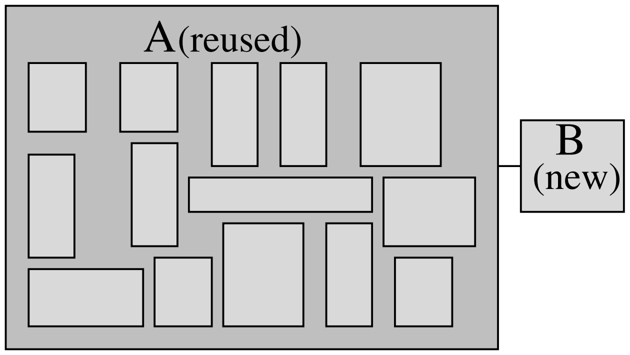

Figure 5. Adding a new component B does not change the relevant behavior of the reused subsystem A under the arbitrary composability property.

on the bus and may increase the communication delay. In turn this can lead to missed deadlines and even to deadlocks which did not exist before. Thus, the entire system has to be reanalyzed, possibly redesigned and reverified even if the new core was a relatively small and insignificant addition.

The main benefit of the arbitrary composability property comes from its implications on design effort and the potential of reuse. This consequence can be formulated as follows [10].

> **Linear Effort Property:** *Given is a set of components and a set of combinators that allow to connect and integrate the components into larger component assemblages. A design process which builds a system from the components and combinators has the linear effort property if a given set of n assemblages $A_1, \ldots, A_n$ can be integrated into a system S by means of the combinators with an effort dependent on n but not on the size of the assemblages:* $\texttt{Ieffort}(n)$. *Thus the total design effort for S is*

$$\begin{aligned}\texttt{Deffort}(S) &= \texttt{Deffort}(A_1) + \cdots \\ &\quad + \texttt{Deffort}(A_n) + \texttt{Ieffort}(n)\end{aligned}$$

This means that the integration effort only depends on the number of components and subsystems but not on their size. Thus, to reuse large components takes as much time as to reuse small components. This means that we can build more and more complex systems by reuse and integration without limit.

Both properties are essentially equivalent. If the arbitrary composability property holds, the integration effort is independent of the size and complexity of the individual components; it only depends on the number of components to be integrated. On the other hand if the system is not arbitrary composable, the design effort will depend on the size and complexity of individual components because they have to be reanalyzed and possibly redesigned.

We argue that arbitrary composability is a central property of a computational model for complex SoC platforms. Even though it incurs some overhead and prohibits the most optimal solutions, we am convinced that the overhead costs are rather limited and the potential gains in terms of predictable composition, ease of reuse, analysis and verification is tremendous. In the next section we introduce the important features of the Nostrum NoC which we use as an example for formulating a MoC with the arbitrary composability property.

4. Nostrum

The Nostrum NoC deploys a regular mesh topology. Figure 6 shows a 4×4 network. Switches are connected to each other and form the network. A network interface (NI) forms the interface between a switch and a resource. A resource is connected to exactly one switch in its north-east direction. A resource can be anything between a simple,

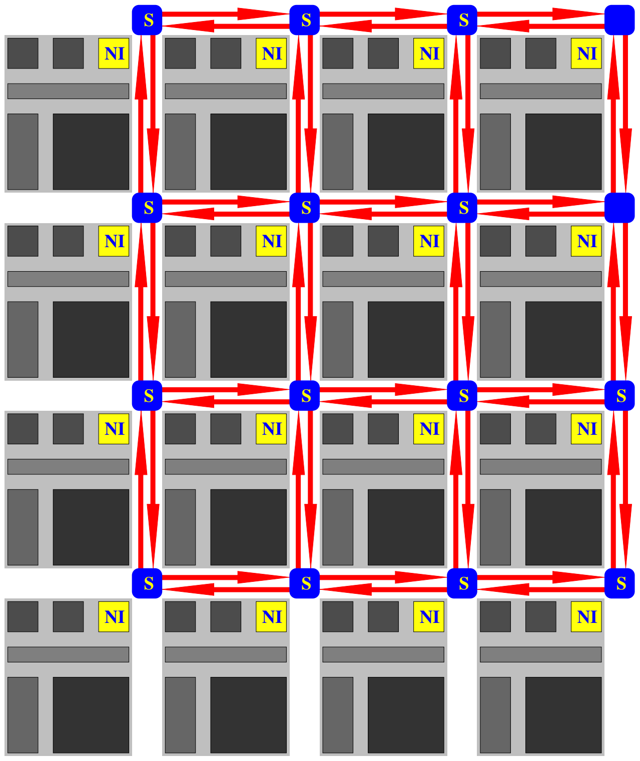

Figure 6. A 4x4 Nostrum NoC.

dedicated hardware block to a local bus-based cluster with a processor, local memory and peripherals. A switch is up to five input and output links; peripheral switches have less. Each link contains 128 wires and a few control lines. Switches use an adaptive routing strategy called deflection routing. A switch tries to send a packet to the direction of its destination. If the corresponding output link is blocked by another packet, the packet is deflected into another direction and effectively moved away from its destination. This means, switches do not temporarily store packets and therefore they have no internal buffers. They only have an input and an output buffer for each link. Each packet is 128 bit wide, contains its target address and is independent of its predecessor and successor packet. This scheme allows for relatively simple, buffer-less switches because it uses the network itself as packet buffer.

The network interface (NI) constitutes the bridge to the resource. It provides more sophisticated services to the resource such as end-to-end flow control and in-order packet delivery.[2] More important for our discussion it offers two different communication quality classes: best effort (BE) and guaranteed bandwidth (GB).

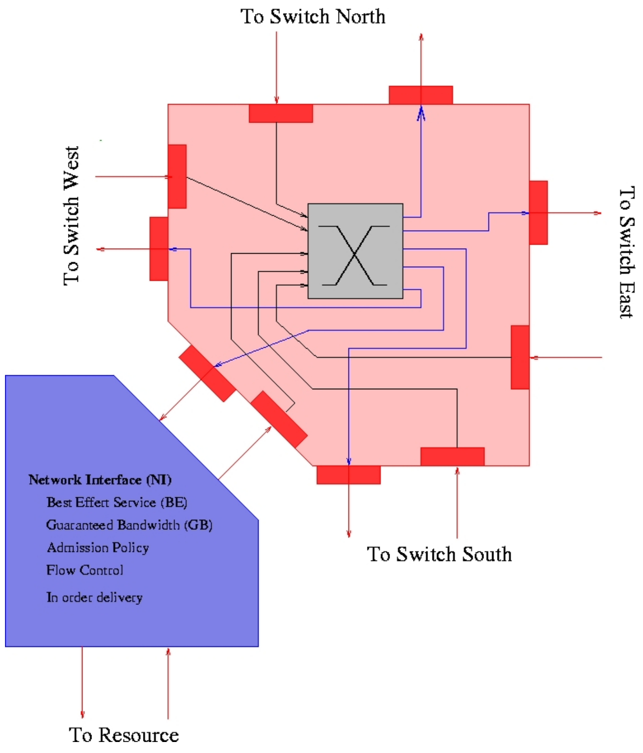

Figure 7. The Nostrum switch and network interface (NI).

Best Effort (BE) packets are buffered in the NI until it is possible to inject them into the network. The NI implements an admission policy that determines when a

[2] In fact, these services are provided by two blocks, called network interface (NI) and resource network interface (RNI). But for the sake of a simple presentation in this article we subsume both into the NI.

new BE packet can be injected. When the concerned switch is fully occupied, i.e. when it receives four packets from the neighboring switches and none of them is targeting the local resource, then obviously no new packet can be injected. But the admission policy is typically stricter than that to keep the load in the network below an acceptable level.

Once the BE packet is in the network, it may be deflected into an unfavorable direction if it competes with another packet for a link. When two packets compete the older packet (higher hop count) wins. If both have the same age, the one which is closer to its destination wins. If both have the same distance to the target resource, the decision is taken randomly. This priority policy ensures that packets are not trapped and all packets eventually leave the network and an upper bound on the packet delay can be established.

Guaranteed Bandwidth (GB) connections are realized with *virtual circuits (VC)* [12]. A VC between two resources **A** and **B** is established by allocating time slots in each switch on the path between **A** and **B**. VC packets always have priority over BE packets. Hence, they can only compete with other VC packets. However, VCs have to be set up in a way that no contention between them can occur. Consequently, a VC packet that has entered the network will never be deflected and will therefore reach the destination in the time defined by the VC.

To make sure a VC packet is able to enter the network, an empty VC container is traveling the VC in a loop (figure 8). When the container passes by the source node, it loads the packet and transports it to the destination along the VC path. In that way it is guaranteed that the VC packet can enter the network at least when the VC container arrives. The VC (which is defined by the switch configurations) and the VC container provide together the GB service.

These services are provided by the network and can be used by the application. It is obvious that the application can overload the network if it is not restricted. The goal is to utilize the network as fully as possible without compromising the quality of the communication services while realizing the arbitrary composability property. This is our objective when designing the MoC for Nostrum.

5. A Nostrum MoC

Assuming we have an existing design based, called **A** on a Nostrum communication network with an application running. The resources use the network to communicate

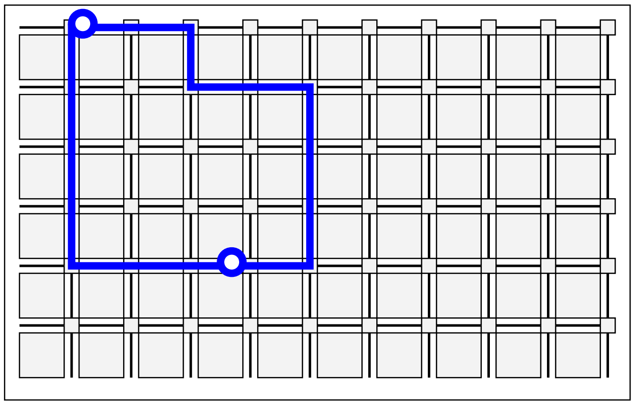

Figure 8. A container loops in a closed virtual circuit (VC) to provide GB traffic.

with each other. We have verified the functionality and we are content with the performance.

We consider two types of composition.

(a) We add one or more resources to the network, which communicate among themselves and with "old" resources. Adding new resources implies adding new switches and communication bandwidth. Before adding these new subnetwork, called **B** to our existing design **A**, we are obliged to verify functionality and performance of **B** including its internal communication before merging it with **A**. Hence, we have only to consider the communication between **A** and **B** and its impact on the network behavior.[3]

(b) We add new tasks or new communication demands in the existing application without adding a new resource.

Assuming that the existing sub-systems have been design and verified properly, both types of composition are reduced to the problem to add new communication demands to an existing and working system.

5.1 GB Composition

The GB traffic based on virtual circuits (VC) is independent of the BE traffic, because whatever the amount of BE traffic in the network, the VC packets will always exhibit the same delay. The BE traffic is completely transparent to the GB traffic. Hence, we can treat the VCs as if there were no BE traffic in the network.

In a given network we have L links, where a link is a unidirectional connection between two switches. In Nostrum two neighboring switches are connected by two links. Hence, in a $n \times n$ network we have $L = 4n^2 - 4n$ links. We denote the load on a link with l_i with $0 \leq i < L$.

[3]In fact, we also have to consider the impact of the new addition to the resources in **A**. But we suggest this is an independent and orthogonal problem that also has to be addressed by relying on a composability property.

Nostrum is a pseudo-synchronous network [14, 13] which, for our purpose can be treated as a synchronous network. A VC essentially allocates a number of links on its path during specific time slots. To simplify explanations and arguments we use a time window as a global reference frame. All time slots are numbered within this window. All VC allocations are repeated in each time window. Let the length of the window be W. A typical value of W would be $2D$ where D is the diameter of the network measured in clock cycles. E.g. In Nostrum with only output registers in all switches the diameter of a $n \times n$ network is $D = 2n - 1$. Thus, a reasonable value of W in a 4×4 Nostrum network is 14.

VCs are defined by the time slots for which they allocate links during the time window W. The load on link i by VC k is given by $v_{i,k}$. If a VC uses only a single container we have $v_{i,k} = 1$ for all the links on the path of the VC. By definition the load on a link must be $v_{i,k} \leq W$. The load of a VC on the entire network is $V_k = \sum_i v_{i,k}$.

The Nostrum MoC must constrain the load that VCs can put on individual links and the network as a whole, which is formulated as follows.

$$\sum_k V_k \quad \leq \quad CG_{\mathrm{VC}} \qquad (1)$$

$$\sum_k v_{i,k} \quad \leq \quad CL_{\mathrm{VC}} \quad \text{for all links } i \qquad (2)$$

If we do not need to reserve bandwidth for BE traffic, we have $CL_{\mathrm{VC}} = W$ and $CG_{\mathrm{VC}} = WL$. However, in all cases where the network is shared by BE and GB traffic, each link must reserve some bandwidth for BE traffic. How much the BE share should be depends on the expected amount of BE and VC traffic and on its spatial and temporal distribution. For non-uniform traffic it may be sensible to set CL_{VC} differently for each link.

Based on the constraints (1) and (2) of the Nostrum MoC the composition rule for adding new VCs to an existing network simply states that all old and new VCs together have to comply with the constraints (1) and (2). This will guarantee that all "old" VCs will not be affected and that BE traffic still has the amount of bandwidth available it expects (see next section). An obvious consequence of this rule is, that it may not be possible to allocate a desired new VC. However, this is expected for any limited resource system. The advantage of formulating the MoC in this way is that a feasibility check can be performed statically at design time.

5.2 BE Composition

Composing BE traffic is by definition more difficult because it is less predictable. If resources are not restricted with respect to the amount of BE traffic they release into the network, then both predictability and the possibility of static analysis will vanish.

We model all BE traffic as *channel* based. When resource **A** sends data to resource **B** we say there is a channel between **A** and **B**. Channel h loads the network with

$$E_h = n_h d_h \delta$$

where n_h is the number of packets **A** injects into the network in the window W, d_h is the shortest distance between **A** and **B**, and δ is the *average deflection factor*, or simply *deflection factor*. δ denotes the average amount of deflection a packet experiences. It is defined as

$$\delta = \frac{\text{sum of traveling time of all packets}}{\text{sum of shortest path of all packets}}$$

Obviously, δ depends on the load in the network.

The load of channel h on the links of all shortest paths between **A** and **B** is

$$e_{h,i} = n_h p_{h,i} \delta$$

where $p_{h,i}$ is the probability that link i is used by a packet of channel h on the shortest path between **A** and **B**. An example of the possible shortest paths between two resources and the corresponding link probabilities is shown in figure 9.

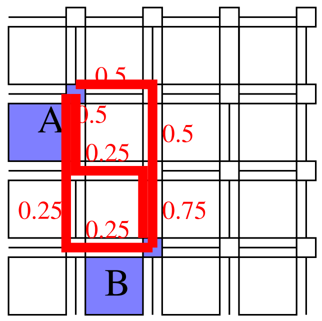

Figure 9. There are three different shortest paths between A and B. The probabilities of packet taking a particular link is depicted in the figure.

Note, that we are not accurate here. When a packet is deflected, it probably gets deflected off the shortest path and would therefore load another link not on any of the shortest paths in the channel. This is not reflected in the formula for $e_{h,i}$. To some extend this is compensated by other channels that load our links even though they are not on their shortest paths, but we have to observe the effects of this inaccuracy and verify that it does not jeopardize our conclusions about the composability of traffic.

Just as for VCs we have to constrain the amount of BE traffic as follows.

$$\begin{aligned}\sum_h E_h &\leq CG_{\mathrm{BE}} \leq LW - CG_{\mathrm{VC}} && (3)\\ \sum_h e_{i,h} &\leq CL_{\mathrm{BE}} \leq W - CL_{\mathrm{VC}} \quad \text{for all links } i && (4)\end{aligned}$$

where (3) gives the global constraint and (4) constrains the load on each link.

For the GB traffic equations (1) and (2) are sufficient to design the traffic if we assume that we plan the traffic statically at design time, or if we go for dynamic VC set-up and tear-down, there is a central agent that can perform a global analysis and make decisions. This assumption may or may not be too limiting for GB traffic, but it is certainly unreasonable for BE traffic, which is more dynamic and can be less well planned. Therefore, we need a decentralized procedure to determine if new BE traffic is allowed to enter the network.

We allocate *traffic budgets* to each resource which define how much traffic may enter the network. These budgets have to be calculated statically at design time and may be recalculated dynamically. Once a resource obtains a traffic budget, it can make an autonomous decision about the network access of a packet. In fact the budget has to be enforced by the network interface.

Let H_r^o be the channels that start at resource r and let H_r^i be the channels that end at resource r. Each resource r in the network is assigned an outgoing traffic budget B_r^o and an incoming traffic budget B_r^i,

$$\begin{aligned}\sum_{h\in H_r^o} E_h &\leq B_r^o\\ \sum_{h\in H_r^i} E_h &\leq B_r^i && (5)\\ \sum_r B_r^o = \sum_r B_r^i &\leq CG_{\mathrm{BE}}\end{aligned}$$

This is essentially the same constraint as in equation (3) but formulated from a resource perspective rather than from a channel perspective. However it allows us to distribute the network capacity over the resources and implement local admission policies in each network interface that enforce the traffic budget constraints.

Both (3) and (5) are *necessary* constraints but they may not be *sufficient* for the proper working of the network because they are global constraints that do not consider local loads on links. There are two alternatives to derive more accurate constraints. One is to follow up on equation (4) and distribute budgets for loads on individual links to the resources. This would lead to an accurate and conservative, hence reliable, boundary on the admitted traffic. However, this strategy is most suitable for deterministic routing where the channel's source and destination addresses unambiguously define all the links on the way. However, for adaptive routing this approach leads to a very involved analysis with a complicated admission to be implemented in the network interface. Therefore we explore here another option.

In equation (3) we noted that $CG_{\mathrm{BE}} \leq LW - CG_{\mathrm{VC}}$. If we would set them equal it means we attempt to utilize all the available bandwidth for BE traffic. This is not very realistic because it tends to drive the network in a saturated mode with poor performance characteristics. We accept this observation and rewrite the inequality as

$$CG_{\mathrm{BE}} = \kappa(LW - CG_{\mathrm{VC}}) \quad \text{with } 0 \leq \kappa \leq 1 \qquad (6)$$

κ gives the margin that we have to account for local overloading of the network; we call it the *global, average, traffic ceiling* or simply *traffic ceiling*.

5.3 MoC Properties

Assuming for a second that we have a way to determine effective and conservative values for the two constants that we used, the deflection factor δ and the traffic ceiling κ, we have obtained a composable traffic model that is our desired MoC. It is defined by equations (1) and (2) with constants CG_{VC} and CL_{VC} for GB traffic and equations (5) with constant CG_{BE} for BE traffic. It states that new traffic can be added to the existing network and traffic constellation if it meets these constraints.

5.3.1 GB Traffic

For the GB traffic latency and bandwidth are precisely defined. A VC k provides the following bandwidth

$$\mathrm{BW}_k = \frac{n_k}{W} \qquad (7)$$

where n_k is the number of packets admitted into the network in each window period W. The maximum latency of packets guaranteed by the VC k is

$$\mathrm{maxLat}_k = \mathrm{maxInit}_k + \mathrm{len}_k \qquad (8)$$

where $\mathrm{maxInit}_k$ is the maximum time between two allocated time slots. If there is only one slot allocated we have $\mathrm{maxInit}_k = W - 1$. len_k is the length of the VC measured in cycles. The average latency is given as

$$\mathrm{avgLat}_k \simeq \frac{\mathrm{maxInit}_k}{2} + \mathrm{len}_k \qquad (9)$$

This is only an approximation since it depends on the distribution of the allocated slots within the window period but it can be made precise if necessary.

5.3.2 BE Traffic

BE traffic latency and bandwidth cannot be predicted as accurately, because they depend on the run-time situation in the network that varies dynamically. Both the avergare latency and the bandwidth depend critically on the deflection factor δ which in turn depends on the traffic load which is bounded by the traffic ceiling κ. However, there is a maximum latency, which can be found as follows.

Since BE packets have different priorities based on their age, there must be a packet with the highest priority in the network. In a network with $N = n \times n$ nodes, the worst case situation is that there are N packets with the highest priority. There cannot be more because at most N packets can be emitted simultaneously. Let the time instance of consideration be t_1 when we have N packets with highest priority in the network. Let's call this oldest generation the set P.

How long does it take for them to leave the network? These packets only compete against each other and when two of them compete for a link, one will win and decrease its distance to the destination, which means that packet has a higher priority. Hence, there will be at least one of these packets that comes closer to its destination in each and every switching cycle. The worst case is that this packet needs D cycles to the destination after t_1; recall that D is the diameter, i.e. the longest shortest path in the network. Thus, D cycles after t_1 there will be at most $N-1$ highest priority packets in the network. We repeat the argument for each of these packets and find that at most ND cycles after t_1 all N packets of the set P have left the network.

What is the worst possible value for t_1 for our N packets? Assume packets P enter the network at time t_0. At that time all packets in the network must be older and there can be at most L packets in the network because L, the number of links, is the maximum capacity. At most after DN cycles the oldest of these packets have left the network as we found above. This means, DN cycles after t_0 there are at most $L - N$ packets in the network older than the packets in P. After at most another DN cycles N more packets will have left the network and this repeats until there are no older packets in the network than those in P, which means we have arrived at t_1. The time between t_0 and t_1 is thus mDN with

$$L - mN = 0$$

We find m as follows.

$$m = \frac{L}{N} = \frac{4n^2 - 4n}{n^2} = 4 - \frac{4}{n} < 4$$

Thus, a bound for $t_1 - t_0$ is $4DN$ and a bound for maximum latency of a packet on any BE channel h is

$$\mathrm{maxLat}_h = 5DN \tag{10}$$

This is a rather high and conservative upper bound. The average latency is always far below it. On a BE channel h it is

$$\mathrm{avgLat}_h = d_h \delta \tag{11}$$

where d_h is the shortest distance on the channel.

The BE bandwidth available to a resource r is limited by the outgoing traffic budget of that resource and by the incoming traffic budget of its communicating partner.

$$\mathrm{BW}_r = \sum_{h \in H_r^o} \frac{n_h}{W} \tag{12}$$

under the budget constraints (5). This means a resource is allowed to set-up new channels if it complies with its own outgoing traffic budget and its communication partner's incoming traffic budget. It also means that the available bandwidth depends on the communication distance; if all partners are close by, the resource has more bandwidth than for long distance communication.

5.4 Deflection factor δ and traffic ceiling κ

We have formulated a MoC for Nostrum that allows us to statically reason about communication performance, traffic, and traffic composition. However, the empirical factors traffic ceiling κ and deflection factor δ have to be determined. δ is used for describing the average latency for BE traffic (11) and κ is part of the budget constraints (5) for describing the available bandwidth to a resource (12).

First we note that δ and κ depend on each other. Exactly how they are related depends on the routing, switching and buffering policies of the network. For deterministic routing the dependency is weak because κ is a network-global parameter while δ is a channel-local parameter. For a given κ it is easy to overload a specific channel with a sharply increasing δ for that channel without violation κ. For an adaptive, non-minimal routing policy the relation is much stronger because the load on an individual BE channel is not confined to a sequence of a few links but spreads over the entire network. Consequently, we have the hope to set a global traffic ceiling κ and derive an average deflection factor δ valid for all BE channels.

Table 1 shows the measured deflection factors for a range of network sizes and emission budgets per cycle B_r^o/W for uniformly distributed traffic. No VC traffic has been considered, hence $CG_{\mathrm{VC}} = 0$. The global traffic budget κLW has been uniformly distributed to the individual nodes and relates to B_r^o according to (5) and (6) as follows.

$$\frac{B_r^o}{W} = \frac{\kappa L}{N}$$

Table 1. Deflection factors δ for various network sizes N and emission budgets per cycle B_r^o/W.

B_r^o/W	16	20	30	40	50	60	70	80	90	100
0.05	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02
0.10	1.04	1.05	1.04	1.06	1.06	1.05	1.05	1.06	1.05	1.05
0.15	1.07	1.08	1.08	1.11	1.10	1.09	1.09	1.10	1.09	1.09
0.20	1.10	1.12	1.11	1.17	1.15	1.15	1.14	1.16	1.14	1.14
0.25	1.14	1.17	1.15	1.26	1.24	1.23	1.22	1.24	1.22	$sat.$
0.30	1.18	1.22	1.20	1.46	1.41	1.36	1.33	1.33	1.29	$sat.$
0.35	1.23	1.28	1.27	1.78	1.65	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.40	1.28	1.36	1.40	1.98	1.84	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.45	1.35	1.54	1.75	1.99	1.85	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.50	1.57	1.98	1.82	1.99	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$

Entries marked with *sat.* in table 1 denote cases when the network is saturated and $\delta \geq 10$. We observe that δ is relatively stable and can be predicted based on the network size and the traffic budget. As a conservative upper bound for δ we propose D_1 values. D_i is a family of normalized delay values defined as follows.

$$1 - 10^{-i} \text{ of all packets } p\text{: } \frac{\text{delay}(p)}{\text{mindelay}(p)} \leq D_i \qquad (13)$$

Thus, 90% of all packets have a delay less or equal D_1. Table 2 gives the D_1 values for the same scenarios as those of table 1. We can see that D_1 is an upper bound for δ and follows it fairly closely as long as the network does not approach its saturation point. Using D_1 as approximation for δ will encourage to set the traffic budgets B_r^o such that the network is operated well below its saturation point.

We believe that we can empirically find D_1 values for a particular network and traffic budgets which provide excellent bounds for the deflection factor δ and consequently also for average latency and available bandwidth assuming we know the traffic pattern sufficiently well. However, this is a strong assumption because in practice it may be very difficult or even impossible to predict the traffic patterns of real applications. Thus, we need to understand how sensible the measurement of D_1 values depend on variations of the traffic scenarios. Table 3 shows deflection factors for an 4×4 network and various traffic scenarios. The second column gives the uniform traffic and equals column two in table 1. Columns three through seven (BitComplement - BitTranspose) represent scenarios where the target address is a bit permutation of the source address as suggested by Dally and Towles [5]. The last column, denoted by BUniform, represents a bursty traffic scenario following the traffic generators proposed by Wang et al. [19] with a factor $b = 0.2$. Somewhat surprisingly the deflection factor of that column is lower than the one with the Uniform traffic. The reason is that the bursts emerging from different resources rarely appear simultaneously and therefore the different traffic streams interfere less with each other than in the Uniform traffic scenario. Our most important observation is however, that the D_1 values of column two in table 2 are bounds with sufficient margins for all δ values in table 3. However, we have to be careful to understand that this is a preliminary finding and there is no guarantee that a traffic scenario of a real application will not behave worse and exhibit deflection factors higher than the D_1 bounds derived from simulations with other traffic models.

6 Discussion

We have described a Model of Computation for Nostrum that is based on two types of traffic, Guaranteed Bandwidth (GB) and Best Effort (BE) traffic. For the GB traffic we have proposed a central planning and resource allocation of all GB channels. This central planning can be done either at design time or dynamically at run-time by a central agent in the network. The properties of the GB traffic is described in section 5.3.1 by equations (7), (8) and (9).

For the BE traffic we suggest to assign traffic budgets to each resource. The assignment of budgets can again be performed at design time or at run-time by a central agent in the network. The network interface ensures that each resource complies with the allocated budget. As long as each resource stays within its traffic budgets the performance properties of the BE traffic are characterized in section 5.3.2 by equations (10), (11) and (12).

The benefit of this approach is that both GB and BE traffic performance is concisely characterized by formulas, can be predictably planned, analyzed, and composed. The disadvantage is that network resources are allocated that may not be fully utilized. If a GB virtual circuit is not fully used, its bandwidth cannot be used by other traffic and it is wasted. If a resource does not fully use its allocated bud-

Table 2. D_1 values for various network sizes N and emission budgets per cycle B_r^o/W.

B_r^o/W	16	20	30	40	50	60	70	80	90	100
0.05	1.12	1.12	1.12	1.15	1.15	1.15	1.16	1.16	1.11	1.11
0.10	1.12	1.12	1.15	1.25	1.23	1.23	1.23	1.27	1.23	1.23
0.15	1.12	1.28	1.30	1.41	1.41	1.36	1.35	1.41	1.35	1.35
0.20	1.36	1.44	1.40	1.46	1.46	1.46	1.46	1.46	1.47	1.55
0.25	1.44	1.44	1.45	1.65	1.64	1.71	1.80	2.35	3.46	$sat.$
0.30	1.44	1.60	1.61	4.65	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.35	1.60	1.61	1.72	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.40	1.60	1.81	6.10	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.45	1.80	3.44	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$
0.50	6.17	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$	$sat.$

Table 3. Deflection factors δ for an 4×4 network and various traffic patterns and emission budgets B_r^o.

B_r^o/W	Uniform	BitComplement	BitReverse	BitRotation	BitShuffle	BitTranspose	BUniform
0.05	1.02	1.01	1.01	1.00	1.01	1.00	1.00
0.10	1.04	1.02	1.02	1.00	1.02	1.01	1.02
0.15	1.07	1.05	1.03	1.00	1.03	1.02	1.03
0.20	1.10	1.07	1.04	1.01	1.05	1.02	1.05
0.25	1.14	1.11	1.04	1.01	1.06	1.02	1.05
0.30	1.18	1.17	1.06	1.02	1.09	1.02	1.07
0.35	1.23	1.24	1.07	1.02	1.11	1.03	1.08
0.40	1.28	1.34	1.09	1.03	1.15	1.03	1.09
0.45	1.35	1.62	1.11	1.04	1.18	1.04	1.11
0.50	1.57	1.62	1.14	1.04	1.21	1.05	1.12

get, other resources are not able to utilize it.

Several variations and optimizations of this approach are conceivable. Traffic planning for both GB and BE traffic can be done purely statically at design time. Alternatively, traffic reallocation can be performed dynamically at run time by a central network agent, e.g. a network operating system. When this is beneficial depends on the dynamics of the application and the overhead involved.

GB traffic could also be planned based on resource budgets just like the way we described for BE traffic. This would mean that virtual circuits (VCs) have to be opened and closed dynamically constrained by the GB budgets of the communicating resources. Similarly, BE traffic could be allocated on a channel by channel basis, just as we did for GB traffic. Which of the four combinations, i.e. (GB resource budget, BE resource budget), (GB channel allocation, BE resource budget), (GB resource budget, BE channel allocation) or (GB channel allocation, BE channel allocation) exhibits the best trade-off depends on the application traffic characteristics and the involved run-time overheads.

Our proposed characterization of BE traffic is based on two empirical parameters, the deflection factor δ and the traffic ceiling κ. Since this option requires that traffic load adaptively spreads out over the entire network it seems to be viable for the adaptive, non-minimal routing strategy of Nostrum. A deterministic routing mechanism probably requires a different approach based on the utilization of individual links. We believe this is possible with comparable results but leave it for the future. However, we expect some differences making one kind of network more suitable for one type of application. Deterministic routing based networks probably allow for MoCs with more accurate performance characterizations of well defined traffic patterns. Adaptive networks are probably superior for applications with less well known or more dynamic traffic patterns.

As a result of this study we believe a MoCs can be developed that allow accurate characterization of communication performance in NoCs. A MoC has to allow for effective planning of traffic. We assign particular significance to MoCs that allow for efficient traffic composition and the analysis of the resulting system performance. We believe this is possible for a wide array of routing, switching and buffering policies but it is mandatory to design the networks communication services such that efficient MoCs can be formulated. Not every arbitrary NoC will lead to a useful MoC abstraction.

Acknowledgment Many ideas of this article have been influenced by numerous debates with members of the System Architecture and Methodology Group at KTH. In particular I found discussions with Zhonghai Lu, Mikael Millberg, Ingo Sander and Johnny Öberg highly inspiring.

References

[1] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *19-th Annual ACM Symposium on Theory of Computing*, pages 305–314, May 1987.

[2] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3–28, March 1990.

[3] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2/3):72–109, 1994.

[4] S. Cook and R. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7:354–375, 1973.

[5] W. J. Dally and B. Towels. *Principles and Practices of Interconnection Networks*. Morgan Kaufman Publishers, 2004.

[6] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing*, San Diego, CA, 1978.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W.H.Freeman and Company, 1979.

[8] P. B. Gibbons, Y. Matias, and V. Ramachandran. The QRQW PRAM: Accounting for contention in parallel algorithms. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 638–648, Arlington, VA, January 1994.

[9] A. Jantsch. Models of embedded computation. In R. Zurawski, editor, *Embedded Systems Handbook*. CRC Press, 2005. Invited contribution.

[10] A. Jantsch and H. Tenhunen. Will networks on chip close the productivity gap? In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, chapter 1, pages 3–18. Kluwer Academic Publishers, February 2003.

[11] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of parallel computation: a survey and synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 61–70, 1995.

[12] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, February 2004.

[13] E. Nilsson and J. Öberg. Reducing peak power and latency in 2-D mesh NoCs using globally pseudochronous locally synchronous clocking. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, September 2004.

[14] E. Nilsson and J. Öberg. Trading off power versus latency using GPLS clocking in 2D-mesh NoCs. In *Proceedings of the International Symposium on Signals, Circuits and Systems (ISSCS)*, July 2005.

[15] J. E. Savage. *Models of Computation, Exploring the Power of Computing*. Addison Wesley, 1998.

[16] R. G. Taylor. *Models of computation and formal language*. Oxford University Press, New York, 1998.

[17] E. Upfal. Efficient schemes for parallel communication. *Journal of the ACM*, 31:507–517, 1984.

[18] P. van Embde Boas. Machine models and simulation. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 1, pages 1–66. Elsevier Science Publishers B.V., 1990.

[19] M. Wang, T. M. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *ICDE*, 2002.

[20] T. Williams. *A General-Purpose Model for Heterogeneous Computation*. PhD thesis, University of Central Florida, Orlando, Florida, USA, December 2000.