2 Communication Architecture

This chapter articulates the role of communication architecture in the design of a parallel system. The various parallel machine designs are all converging to the point where the nodes of a parallel system are essentially complete sequential computers that are interconnected by a low-latency packet-switched network. The difference between the approaches is how and at what level this network is integrated into the node architecture.

The communication architecture describes extensions made to a sequential architecture to turn it into a parallel computer architecture. In this sense, it focuses on the integration of communication into the overall architecture. The brief review of existing systems in Section 1.3 indicated the current tendency to fuse the architectural layers and address high-level problems low in the hardware and vice-versa. An important aspect of the development of Active Messages is to reexamine the layering embodied in existing parallel systems and to redistribute the responsibilities to arrive at more versatile and efficient parallel systems.

Placing the primary focus on the layering of the communication functions results in a a novel approach to parallel computer architecture quite different from the traditional ones which concentrate on the examination of alternatives at each level. Section 2.1 contrasts this new “vertical” approach to the more common “horizontal” classifying approach. It relates the notion of communication architecture to that of conventional computer architecture and arrives at the layering model used in this dissertation. Section 2.2 provides an overview of the Active Messages communication architecture and its layers. Section 2.3 discusses the set of performance metrics used throughout the dissertation to evaluate communication architectures.

2.1 A Vertical Approach

The traditional approach to communication in parallel architectures focuses on classifying alternatives. One of the oldest approaches is Flynn's taxonomy [Fly66] which describes alternatives in the number of instruction and data streams. To this Kuck [Kuc78] adds alternatives in the number of execution streams. Treleaven [Tre85] focuses on the variety of control mechanisms (control driven, pattern driven, demand driven, data driven) and of data mechanisms (private memory and shared memory). Almasi and Gottlieb [AG89] focus on alternatives in interconnection networks, in message passing designs, and in shared-memory communication models. Hwang [Hwa93] discusses the alternatives in interconnects, cache coherence mechanisms, message passing mechanisms, latency-hiding techniques, and so forth.

While such horizontal[1] approaches illustrate the variety of design choices nicely, they lead to a mind-set in which decisions taken at each level are independent from those at other levels. For example, after a discussion of the various network routing algorithms it is tempting to ask which one is best. Yet, such a question is meaningless without considering the design choices at many other levels: the network topology, the message length, the channel width, and the reliability and ordering properties required by the communication model all interact with the routing algorithm in significant ways.

This focus on individual layers does not help the architect decide how to coordinate the layers to bridge the gap from the network topology to the development of compilers for high-level parallel programming languages. For example, what should be exposed by the instruction set architecture and what should be hidden in the implementation? What abstractions are appropriate to expose to the language system? What is the role of the operating system in communication? Resolving these issues requires a different, vertical, perspective to parallel computer architecture, in which one may examine how the different layers of functionality fit together to form an efficient whole.

Defining a layering model which represents a hierarchy of interfaces and optimizing the assignment of responsibilities to the various layers is key to the RISC design process. For example, the Stanford MIPS system design revolves around four layers: the high-level language compiler, the assembly language, the machine language, and the micro-architecture. An important aspect of the design process was to decide at which level to address which issues. For example, register allocation requires high-level information and occurs in the compiler while the avoidance of pipeline hazards is the responsibility of the assembly-to-machine language translator.

The following subsections develop the layering model for communication that is used as a framework throughout the dissertation. The first section reviews computer architecture terminology which is often taken for granted. While, for example, the notion of an instruction set architecture is well established in sequential computer architecture, its equivalent in parallel architecture is less well defined. To date, little care has been taken to separate the issues pertaining to a family of parallel machines from those particular to a given implementation. In addition, the instruction set architecture is usually considered as defining the boundary between hardware and software which is not necessarily accurate in the case of parallel computing given the relative infant state of hardware.

2.1.1 Computer architecture in review

Patterson and Hennessy [PH90] define instruction set architecture (ISA) as referring to "the actual programmer-visible instruction set". They further use the term of *computer organization* to refer to the "high-level aspects of a computer's design, such as the memory system, the bus structure and the internal CPU design" and the term *computer hardware* is used for the "specifics of a machine". Shown schematically in Figure 2-1, *computer architecture* is finally "intended to cover all three aspects of computer design [e.g., ISA, organization, and hardware]". The term computer architecture thus refers to the sys-

[1]. Think of the different levels of abstraction along the vertical axis and the alternatives at each level along the horizontal axis.

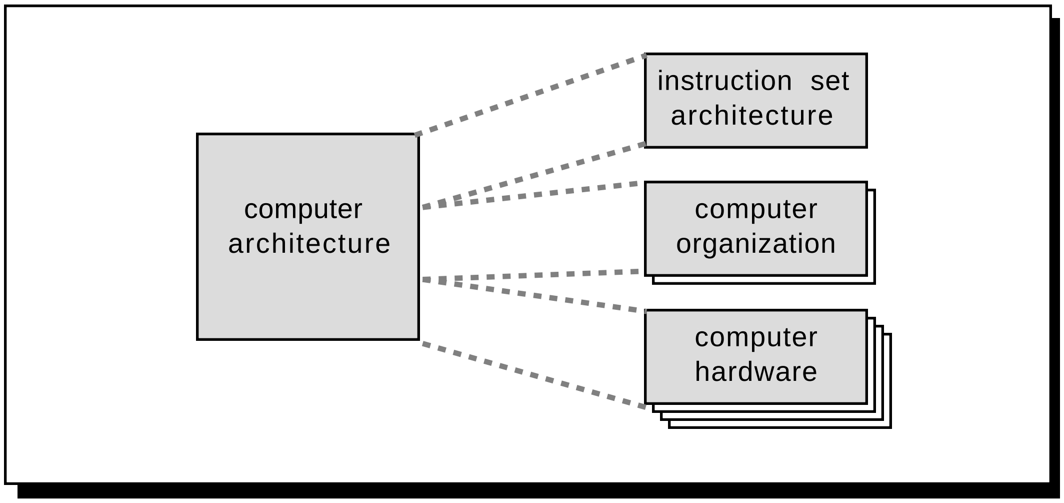

Figure 2-1: Patterson and Hennessy's definition of computer architecture.

Instruction set architecture refers to the actual programmer-visible instruction set, the term *computer organization* to the high-level aspects of a computer's design, such as the memory system, the bus structure and the internal CPU design, and *computer hardware* is used for the specifics of a machine. A single instruction set architecture can be implemented using a number of distinct computer organizations, each of which can in turn be implemented with differing hardware.

tem design including the interface between software and hardware (the original meaning of *computer architecture* as defined by the IBM 360 designers) and the actual design of a computer.

Applied to parallel computing, Patterson and Hennessy's concept of ISA would include the state and the operations relevant to communication, given that these are visible to the programmer. What is less clear, however, is what aspects of communication are "visible to the programmer". For example, is the network topology visible to the programmer? Or is it similar to a cache and considered to pertain to a computer's organization and not to its instruction set architecture? The definition, in addition, seems overly focused on the notion of an instruction, given that the ISA includes other aspects as well.

The definition of computer architecture used by the IBM 360 designers, depicted in Figure 2-2, is more precise in these respects. Translated into today's terminology (they used the term computer architecture to refer to the instruction set architecture) it defines instruction set architecture as being

> "[...] the structure of a computer that a machine language programmer must understand to write a correct (timing independent) program for that machine"

which makes quite clear that issues beyond the instruction execution are included. This is particularly important in a communication architecture where, for example, deadlock is not necessarily directly related to instruction execution yet must be understood by the machine language programmer.[2]

The two points which become clear through the discussion of the various definitions of computer architecture are that

- the interface to the machine language programmer, although typically called the *instruction* set architecture, encompasses more than just the definition of the instruction set, and
- aspects of communication are found at all levels of computer architecture.

2. While network routing is designed to be deadlock-free, it assumes that processors accept incoming messages at all times, even while attempting to inject a message into the network. If the programmer violates this assumption, the network may deadlock.

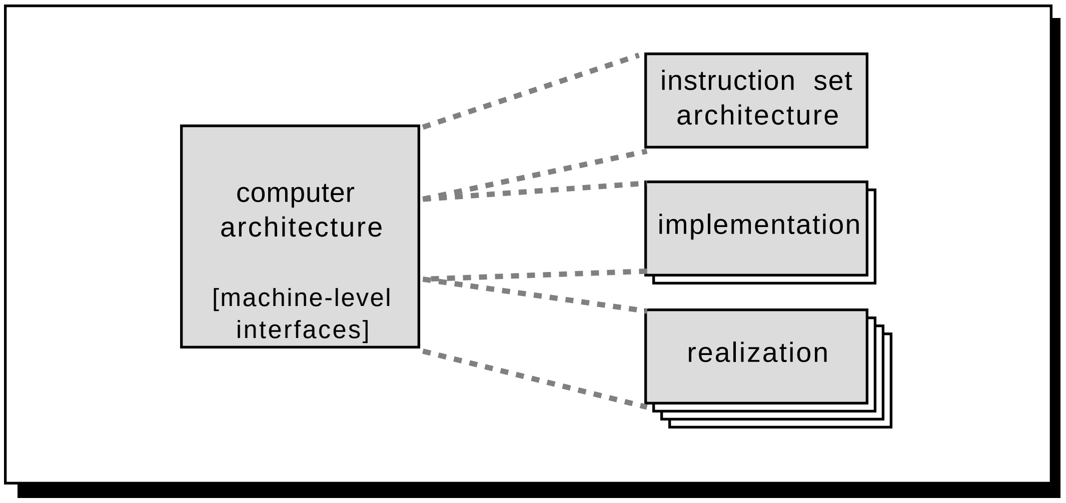

Figure 2-2: IBM 360 designers' definition of computer architecture.

While the distinction between the instruction set architecture and the set of instructions is minor in the case of most modern micro-processors, the term "instruction" becomes confusing in the context of communication. Due to the still infant state of parallel architectures, the vast majority of parallel instruction set architectures define at most a set of communication library functions or kernel traps and only very few have communication *instructions* proper.

2.1.2 Communication layers

Figure 2-3 summarizes how communication enters into computer architecture: it encompasses all aspects of parallel computer architecture that are related to communication and is divided into three levels, namely the communication architecture, the communication micro-architecture, and the communication hardware.

The full layering used in this dissertation adds two levels on top of the communication architecture which represent the software run-time layers of a parallel program. Figure 2-4 shows the following five levels at which communication is defined or used:

- The *communication hardware* level is typically specific to a given machine model and deals with logic design details of the implementation.
- The *communication micro-architecture* describes the organization and operation of the function units involved in communication.
- The *communication architecture* abstracts the implementation specific details into the operations that are visible to the machine language programmer or compiler. In principle, the architecture is implemented in hardware, although it may contain micro-code sequences or even low-level system software in the style of the DEC Alpha PAL code [Sit92].
- The *communication model* defines the application programmer's view of how processors communicate. It is generally embodied in the communication semantics of a parallel language or defined as an extension to a sequential language. The communication model is implemented in a run-time substrate (or through the code sequences generated by the compiler) using the primitives provided by the communication architecture.
- At the *application layer* communication is high-level and machine independent.

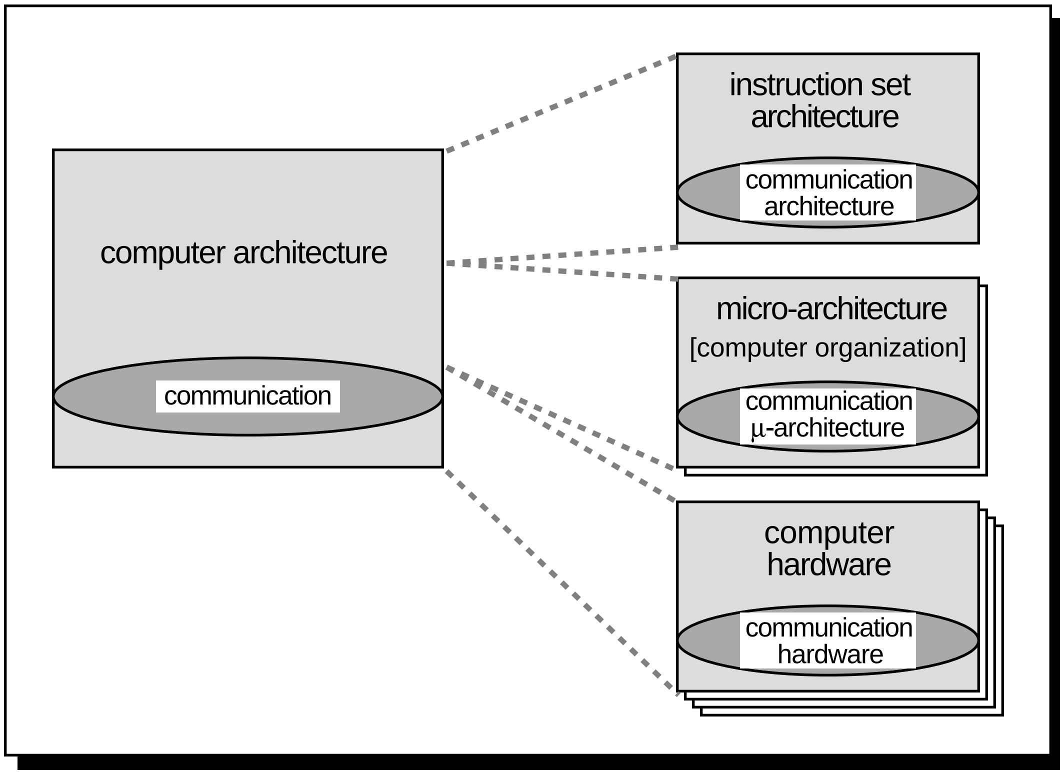

Figure 2-3: Communication architecture as facet of computer architecture.

2.1.3 Aspects of a communication architecture

The communication architecture encapsulates the details of the micro-architecture and defines the programmer-visible communication operations (or even instructions). The communication architecture is typically implementation independent and is carried forth from one model of a parallel computer family to the next.

The primary role of the architecture is to interface between the computer designer and the user. In the traditional sense it represents the boundary between the hardware and the software, but from a RISC design point of view it separates the concerns of the designer and the user without compromising versatility or efficiency. A good architecture will enable high-level optimizations while hiding enough of the μ-architecture to allow for different implementations (with varying price-performance trade-offs) to present the same architecture.

While the communication architecture is conceptually part of the hardware, it is not necessarily implemented fully in hardware. In particular, most older micro-architectures provide little or no support for extending the user protection boundaries across the network. This means that, as undesirable as it is, these architectures offer no alternative but to pass all communication through a privileged software layer in the kernel enforcing the protection.

In order to define a specific communication architecture the following three major aspects of communication need to be specified: (i) the data representation of messages, (ii) the operations to send and receive messages, and (iii) the events signalled by the network to the processor. Table 2-1 presents a detailed template for the definition of a communication architecture.

As an example, the Intel NX send&receive communication architecture defines a number of message sending and receiving operations, a simple message format (destination node, tag, and word-aligned

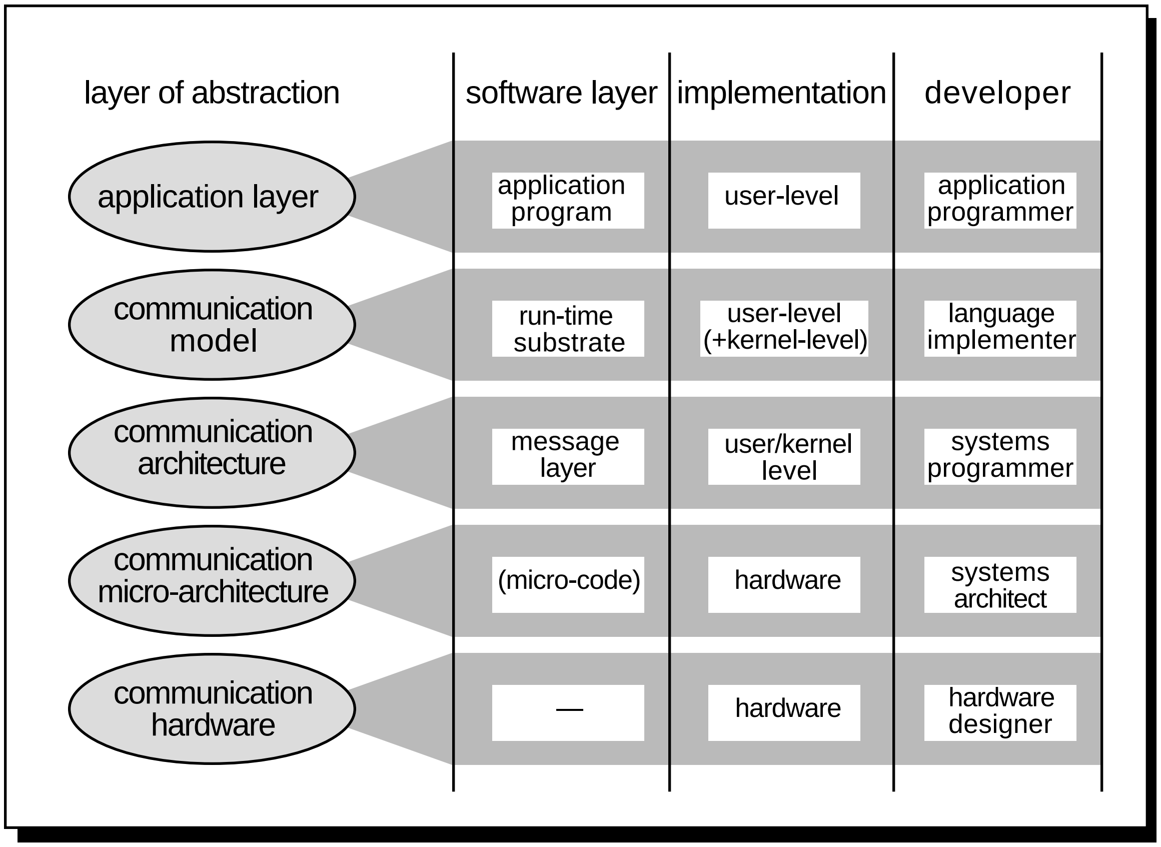

Figure 2-4: Communication architecture layers.

message representation	format	The format of messages including destination address and other header information, maximum length, data format and alignment.
	placement	The placement of messages in the storage hierarchy as expected by send operations and as delivered on reception.
communication operations	send	The operations used to inject messages into the network.
	receive	The operations used to extract messages from the network.
communication events	reception	Events signalling the arrival of a message.
	send completion	Events signalling success or failure of message injection.
	synchronization	Synchronizing events with the ongoing computation, in particular to create critical sections.

Table 2-1. Aspects of a communication architecture.

structure	Network topology	The topology of the network interconnecting processing nodes.
	Network channels	The characteristics of the links connecting processing nodes into the network (and interconnecting routing nodes, if applicable) including: the channel speed and width, flow-control on each link and flit size.
	Message format	The format of messages recognized by the hardware.
	Network volume	The amount of buffer space built into the network which determines the number of messages each processor can pipeline through the network.
operation	Transmission mechanism	Mechanism by which data is moved into the network or extracted from the network. For example, DMA engines or explicit load/store accesses to FIFO queues.
	Network status and events	Status provided by the network interface and, if applicable, processor interrupt or event dispatch mechanisms.
	Routing	Properties of the network routing visible to the message layer. Includes deadlock and livelock avoidance issues.
protection	Protection mechanisms	Mechanisms to prevent unauthorized access to the network from user-level.
	Network context	Network state to be saved and restored on a context switch. Mechanisms to save and restore the state.

Table 2-2. Aspects of a communication micro-architecture.

memory buffer) and hides all network operation and state. It is available on all recent Intel IPSC multiprocessors whose respective communication micro-architectures differ dramatically.

2.1.4 Aspects of a communication micro-architecture

The communication micro-architecture describes the low-level organization of the components performing communication in a parallel machine. It is generally specific to a particular implementation and many of its aspects are not visible to the software, possibly with exception of the kernel.

The major aspects defined as part of the micro-architecture are the structure of the interconnect, the network operation, and the protection mechanisms. Table 2-2 contains a list of the details of the communication micro-architecture that are relevant to the design of the communication architecture.

2.1.5 Glossary

It is quite difficult to refer to the various hardware and software components at each level unambiguously and many appropriate common terms such as “user-level” do not have a universally accepted meaning. For this reason, Table 2-3 defines the terminology used in this dissertation.

TERM	DEFINITION
Protection levels	
Privilege-level, kernel-level	Synonyms, the privileged protection level of software execution, typically called supervisor level or system level.
User-level	Non-privileged protection level of software execution, e.g., level subordinate to the kernel-level.

Table 2-3. Glossary of terms.

TERM	DEFINITION
Software layers	
Kernel, operating system	Software provided by the vendor executing at privilege-level. Kernel usually refers to the software on each node and operating system refers collectively to the kernels on all nodes (and host node, if applicable).
Message layer	Software implementing the communication architecture.
Run-time substrate	Library providing machine independent abstractions used in the implementation of a programming language.
Communication operations	
Comm. operation	Term used at the communication model level to refer to the operations used to perform inter-processor communication, for example, to fetch a copy of a remote data structure.
Comm. primitive	Basic operations used to perform inter-processor communication and provided at the communication architecture level, for example, the Active Messages send primitive.
Comm. mechanism	General term for inter-processor communication techniques used. Refers to the mechanisms implemented by a collection of primitives.
Parallel programs	
Process	Independent user-level thread of control running on one node, e.g., the traditional sequential computing meaning of the word.
Parallel process	Collection of processes, each running on one node of a multiprocessor, and part of the same parallel program execution.
Parallel program	Text of a program for a multiprocessor, sometimes also used as synonym for parallel process.
Parallel processor, multiprocessor, multicomputer	Synonyms for parallel computer. Some classifications [Fox88] distinguish multiprocessors from multicomputer and use the former to refer to machines with processor nodes connected to memory nodes via a network and the latter for machines with processor-memory nodes interconnected by a network. Given that the vast majority of parallel computers today are multicomputers under that classification but that the standard term used for parallel machines is multiprocessor this distinction is not used in this document.

Table 2-3. Glossary of terms. (Continued)

2.2 Active Messages Communication Architecture

The Active Messages communication architecture is based on four observations:

- all micro-architectures communicate using simple messages and cause a small amount of processing to occur at the remote end to handle the message arrival,
- the dispatch on the first word of a message used in message driven architectures is very versatile (as long as it occurs at user-level),
- buffering and scheduling must be kept to a minimum to achieve a lean, fast communication architecture, and
- the majority of systems include a run-time substrate which adapts the communication architecture functionality to the needs of the application or language.

The central mechanism in Active Messages virtualizes the notion of a message interrupt handler found in most communication micro-architectures to user-level:

> Each message contains at its head the address of a user-level handler which is executed on message arrival with the message body as argument. The role of the handler is to get the message out of the network and into the computation on the processing node. The handler is executed "at interrupt time" and must execute quickly to completion.

This communication mechanism:

- allows construction of higher-level communication protocols by providing appropriate remote services through handlers,
- integrates arriving messages into the computation allowing transfer of data directly between application data structures and interaction with the scheduling of computation,
- tolerates communication latencies by continuing computation while communication takes place, and
- provides general processor-to-processor communication using messages with minimal interpretation imposed by the hardware.

Active Message handlers do not perform computation proper, they "only" incorporate arriving messages into the ongoing computation on the node or provide a simple remote service and send back a reply message. This restriction on handlers, as well as further restrictions described in Chapter 4, are necessary to keep the cost of handler execution low.

The contribution of Active Messages relative to message driven models is to determine the right set of restrictions imposed on handlers such that on the one hand handlers can be executed cheaply and on the other hand they have enough "power" to serve as building blocks for higher communication operations.

2.3 Metrics for Communication Performance

A major milestone in the development of RISC instruction set architectures was the definition of quantitative metrics for evaluating design options. The difficulties in defining metrics for communication come from the fact that a communication operation involves multiple units that operate concurrently (e.g., processors, network interfaces, network routers). In order to be able to attribute costs accurately it is important to have a precise model of communication in which the contribution of each unit can be identified.

This dissertation uses the LogP model of parallel computation for this purpose. The following subsection describes the LogP model from the point of view of an architect interested in using the model to understand and characterize machine performance. The LogP model was originally presented in [CKP+93] in a more general algorithm designer's view. Subsection 2.3.2 defines a few additional simpler metrics which are commonly found in the literature.

2.3.1 The LogP model of parallel computation

The LogP model of parallel computation [CKP+93] is intended to characterize machine behavior realistically to serve as a basis for developing fast, portable parallel algorithms. At the same time is captures the machine model against which the algorithm developer optimizes and thereby offers guidelines to machine designers.

The model decomposes a parallel machine into three types of components: processors, network interfaces, and a network, as shown in Figure 2-5. Each component operates autonomously and interacts with other components. The cost of these interactions is captured by the model parameters[3] as follows:

L: an upper bound on the *latency*[4], or delay, incurred in communicating a message containing a word (or a small number of words) from its source module to its target module.

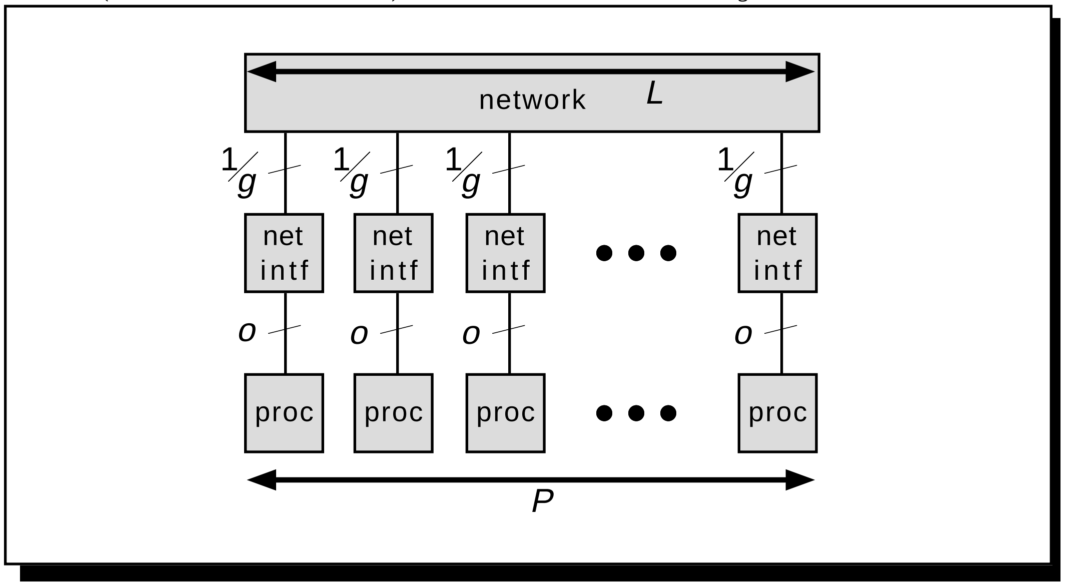

Figure 2-5: LogP components of a parallel machine.

[3] The parameters L, o, and g are measured as multiples of the processor cycle.
[4] The latency experienced by any message is unpredictable, but is bounded above by L in the absence of stalls. Because of variations in latency, the messages directed to a given target module may not arrive in the same order as they are sent.

o: the *overhea*d, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform other operations.

g: the *gap*, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor. The reciprocal of g corresponds to the available per-processor communication bandwidth.

P: the number of *processor/memory* modules. Local operations are assumed to take assume unit time, namely one cycle.

$\lceil L/g \rceil$: the network *capacity* of the network which limits the number of messages can be in transit from any processor or to any processor at any time to $\lceil L/g \rceil$. If a processor attempts to transmit a message that would exceed this limit, it stalls until the message can be sent without exceeding the capacity limit.

Abstractly, these parameters specify the communication delay, and the efficiency of coupling communication and computation. Note that the model does not describe the structure of the network—only its performance characteristics.

2.3.2 Simple metrics

The following few simple metrics complement the LogP model and are used in particular to handle long messages.

Clocks Per Message [CPM]: processor clock cycles taken to send and receive one message.

Pipeline throughput [R_∞] and half-performance point [$N_{1/2}$]: R_∞ is the communication bandwidth achieved with infinite-length messages and $N_{1/2}$ is the message length needed to reach one-half of R_∞. This definition parallels the use of R_∞ and $N_{1/2}$ in measuring the performance of vector computers.

Communication overhead [α] and bandwidth [β]: (also “start-up cost” and “per word cost”) the communication overhead is the time taken by the processor to send and receive a zero-length message and the communication bandwidth measures the number of bytes that cross the network interface per second. The time spent transmitting a message is often modeled as $T = \alpha + l\beta$ where l is the message length in bytes. Note that this ignores the time of flight through the network, which is negligible on many machines given that α is comparatively very large.

2.4 Summary

Applying the RISC design strategy to communication architecture requires a fresh perspective on the various components involved. The predominant view focuses horizontally on the numerous alternatives at each level which results in a tendency to seek the “best” solution at each level, often at the expense of optimizations across multiple layers of abstraction.

The vertical approach proposed in this dissertation focuses on the communication layering model and promulgates the view that optimizing the role of each layer with respect to the other layers is an important part of the design process of a parallel system.

The layering model used in this dissertation establishes the communication architecture as the interface between the computer designer and the user. Its role is to separate the respective concerns without compromising versatility or efficiency. The communication architecture builds on the communication micro-architecture which extends the processor micro-architecture to encompass the structure of the network at the functional unit level. While the micro-architecture is machine dependent, the communication architecture is typically expected to remain compatible from one model of a computer family to the next.

One of the layering model’s most important characteristics is the reliance on a run-time substrate to implement the communication model used in parallel languages or by application programmers. This means that the communication operations seen by the programmer need not be reflected directly in the communication architecture. Instead, the communication architecture must provide the right primitives such that a compiler (or a library writer) can translate the communication primitives found in high-level parallel languages efficiently into the available primitives. This adoption of a high-level parallel language perspective focuses the evaluation of communication architectures on purely quantitative metrics.