# Evaluating the Hyperbolic Model on a Variety of Architectures

Ion Stoica, Florin Sultan, David Keyes *

Department of Computer Science,
Old Dominion University, Norfolk VA 23529-0162, USA
e-mail: {stoica, sultan, keyes}@cs.odu.edu

**Abstract.** We illustrate the application of the hyperbolic model, which generalizes standard two-parameter dedicated-link models for communication costs in message-passing environments, to four distributed-memory architectures: Ethernet NOW, FDDI NOW, IBM SP2, and Intel Paragon. We first evaluate the parameters of the model from simple communication patterns. Then overall communication time estimates, which compare favorably with experimental measurements, are deduced for the message traffic in a scientific application code. For transformational computing on dedicated systems, for which message traffic is describable in terms of a finite number of regular patterns, the model offers a good compromise between the competing objectives of flexibility, tractability, and reliability of prediction.

## 1 Introduction

Most communication models are based on an empirically inferred linear dependence of the time needed to send a message between two communicating parties on the size of the message. For example, various hardware and software overheads in a parallel environment that are modeled by a fixed component, independent of the message size, and by a variable component, proportional to the message size, are identified in [1]. However, such models (with constant coefficients) cannot accommodate contention in a general fashion. Schemes for partially avoiding contention in routing architectures (e.g., a hypercube in [5]) and for obtaining probabilistic guarantees for propagation times have been proposed, but the problem of quantifying the effect of coexisting messages over the same link on the end-to-end communication performance requires more attention.

The hyperbolic model [4] is a variation on the two-parameter models. Its main goal is to address in a uniform way the modularity increasingly present in modern parallel computing environments, where a message path between two communicating parties crosses multiple processing modules having clearly defined

* This author's work was supported in part by NSF grant ECS-9527169, and by NASA contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

interfaces and distinct functionality. If the twin parameters of every module on a message path are known (either by measurement or functional specification), the hyperbolic model allows them to be combined by a set of simple rules into a single pair of end-to-end parameters. In contrast to models that attempt to globally characterize communication costs independently of data paths, the modular hyperbolic representation is data-driven. It can take advantage of knowledge of connectivity and component parameters along the communication paths to adapt the parameters to specific patterns of communication.

## 2 The Hyperbolic Communication Model

*Given a set of source nodes S, a set of destination nodes D, and a set of messages M in a parallel processing environment such that: (1) every message in M is sent by a node in S to a node in D, (2) every node in S sends at least one message and all messages it sends are in M, and (3) every node in D receives at least one message and all messages it receives are in M, our goal is to estimate the transmission time for every message in M.*

This simply described task is rendered difficult in practice by the multilayeredness of a communication network, by the possibility of message contention, and by message packetization. A message can be latency-bound or bandwidth-bound, depending upon its size and packet granularity, and the layer of the network that is "critical" can shift as message size varies, since each layer may have different latency and bandwidth characteristics. In systems with message contention for network paths, the *effective* latency and bandwidth seen by a given message can be functions of the other messages present. This paper describes a means of deriving just such an effective overall pair of latency and bandwidth parameters by algebraic combination rules of component-wise parameters.

The sets $D$, $S$, and $M$ determine the state of the communication system, which is represented as a directed graph called a *communication graph* ($CG$). A $CG$ has two types of nodes: terminal nodes and internal nodes. The terminal nodes represent the end processes that initiate the sending and receiving of data. Between any pair of terminal nodes the data is passed in messages of various sizes. An internal node or *Communication Block* ($CB$) is an abstract module that embeds all the functions performed by the communication protocols in one or more layers of software and hardware, in order to deliver data from source to destination. A $CB$ manipulates data in units of limited size, called *packets*. Passing a message to a $CB$ may result in splitting it into packets. We say that two or more $CB$s are *dependent* if they share a common resource and therefore only one of them can process data at a given moment, and *independent* otherwise. For example, two $CB$s running on different processors are independent, while if they run on the same processor they are dependent.

The most important measure characterizing a $CB$ is the time required to process a message of size $x$, called the *total service time*. We consider that the packet processing time has two components: a *fixed service time* that is independent of the packet size (e.g., the overhead associated with memory management,

interrupt processing and context switching, the propagation delay) and an *incremental service time* that is proportional to the packet size (e.g., data movement between different protocol layers, building and verifying of the CRC or checksum, packet transmission on the communication network).

Consider a $CB$ characterized by the following parameters: the maximum packet size $p$ (bytes), the fixed service time per packet $a$, and the incremental service time per byte $m$. The total service time $t$ for a message of size $x$ is given by

$$t(x; a, m, p) = a\lceil \frac{x}{p} \rceil + mx, \tag{1}$$

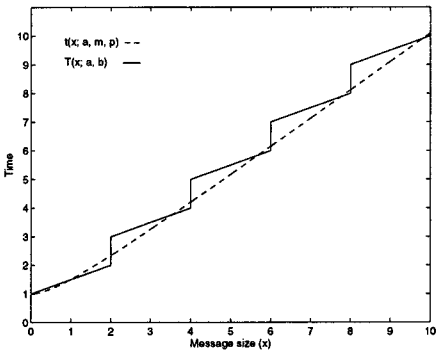


**Fig. 1.** *The total service time t(x;a, m, p) versus the continuous function T(x; a, b) used to approximate it (a = 1, m = 0.5 and p = 2).*

where $\lceil x/p \rceil$ is the number of packets of maximum size $p$ being processed. We approximate the total service time with

$$T(x; a, b) = \frac{a^2}{a + bx} + bx, \tag{2}$$

where $b \equiv a/p + m$ (see Fig. 1). This is the equation of a hyperbola in the $(x, t)$ plane, hence the name of the model. The improvement of (2) over a linear latency ($\alpha$) / reciprocal transfer rate ($\beta$) model, $T(x; \alpha, \beta) = \alpha + \beta x$, is not so much in the fit of a continuous curve to the sawtooth form of a packetized transmission, but in the analytical simplicity with which the parameters $(a, b)$ for a $CG$ may be derived in terms of its elemental $CB$s, as shown by the four combination rules below. Using $T_i$ to estimate the total service time required by $CB_i$ to process a message of a given size, we derive rules for reducing $n$ $CB$s interconnected in various structures to a single equivalent $CB$, with service time $T(a_1, b_1, a_2, b_2, \ldots, a_n, b_n)$. Evaluating the reduced $CG$ at extreme limits of message size and number of processors permits extraction of the salient parameters for the individual $CB$s. A detailed discussion motivating the form of (2) and the combination rules is available in [4].

**Serial Interconnection.** We say that $n$ communication blocks $CB_i$ $(1 \leq i \leq n)$ are serially interconnected with respect to a message $m$ if every packet of $m$ is processed sequentially by every $CB_i$.[2]

[2] Notice that this definition does not imply that a message is processed *in its entirety*

**Rule 1** *Given n serially interconnected communication blocks* $CB_i(a_i, b_i)$, $(1 \leq i \leq n)$, *this structure is equivalent to a single communication block* $CB(a_I, b_I)$ *(for independent blocks) or* $CB(a_D, b_D)$ *(for dependent blocks), where:*

$$a_I = a_D = \sum_{i=1}^{n} a_i; \quad b_I = \max\{b_1, b_2, \ldots, b_n\}; \quad b_D = \sum_{i=1}^{n} b_i \,.$$

**Parallel Interconnection.** We say that $n$ communication blocks $CB_i$ $(1 \leq i \leq n)$ are parallel interconnected with respect to a message $m$ if any packet of $m$ can be processed by any $CB_i$. Assuming that the packets are processed such that the total service time of the message is minimized, we have the following:

**Rule 2** *Given n parallel interconnected communication blocks* $CB_i(a_i, b_i)$, $(1 \leq i \leq n)$, *this structure is equivalent to a single communication block* $CB(a_I, b_I)$ *(for independent blocks) or* $CB(a_D, b_D)$ *(for dependent blocks), where:*

$$a_I = a_D = \min\{a_1, a_2, \ldots, a_n\}; \quad b_I = (\sum_{i=1}^{n} \frac{1}{b_i})^{-1}; \quad b_D = \min\{b_1, b_2, \ldots, b_n\} \,.$$

**Concurrent Processing.** Next we analyze the general case in which a *CB* simultaneously receives for processing $n$ messages $m_1$, $m_2$, ..., $m_n$ of sizes $x_1$, $x_2$, ..., $x_n$. Since we cannot tell exactly when a particular message $m_i$ is processed, we consider the time required to process $m_i$ being bounded by the time required to process all messages, i.e., $m_i$ is the last message being processed.

**Rule 3** *A communication block* $CB(a, b)$ *that processes n messages* $m_1$, $m_2$, ..., $m_n$ *of sizes* $x_1$, $x_2$, ..., $x_n$, *respectively, is equivalent to a structure of n independent communication blocks* $CB_1(a_1, b_1)$, $CB_2(a_2, b_2)$, ..., $CB_n(a_n, b_n)$, *where every* $CB_i$ *processes the message* $m_i$ *and has parameters:*

$$a_i = na; \quad b_i = b \cdot \frac{\sum_{i=1}^{n} x_i}{x_i} \,.$$

**General Reduction Rule.** The previous reduction rules are based on the assumption that the communication graph is identical for both small (packet size) and very large messages. Although this is true for many cases, for complex communication patterns this assumption is no longer valid (see the example of a tree-based broadcast in [4]). We therefore have the following general reduction rule, which interpolates hyperbolically between limiting cases:

**Rule 4** *Given two terminal nodes s and d such that s sends a message m of size x to d, then the total service time for the message m is given by Eq. (2), where a is the service time when sending a small message* $(x \to 0)$, *while b is the service time per data unit when sending a large message* $(x \to \infty)$.

---

by one *CB* and only after that by the next *CB*. In fact, if the message is larger than the maximum packet size and the *CB*s are independent, as soon as $CB_1$ delivers a packet, $CB_2$ can start to process it.

## 3 Communication Parameters

In principle, one can determine *CB* parameters $a$ and $b$ by considering the hardware characteristics of the computation nodes and the communication network (e.g., the processor speed, the memory access time, the internal bus speed etc.) and the communication protocol implementation details (e.g., the number of times a data buffer is copied while passed through various protocol layers, the algorithms used to compute checksums etc.). Although this approach appears to allow accurate evaluation of *CB* parameters, it is hard to apply it in practice due to software and hardware heterogeneity, and due to difficulties in determining the aggregate latency and bandwidth in a complex layered communication architecture, such as the ones embedded in the general purpose operating systems running on the processing nodes.[3]

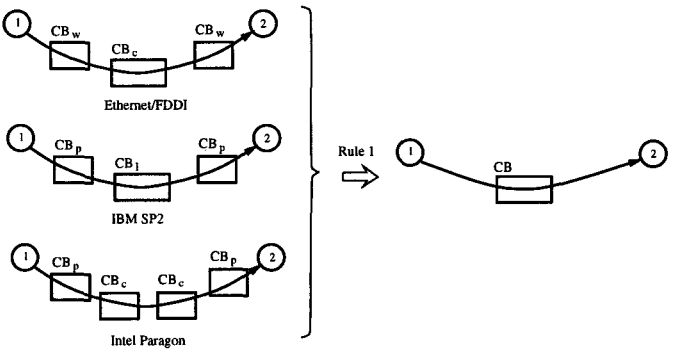


**Fig. 2.** *Communication graphs and their reduction to the equivalent CBs for sending one message between two processors (Pattern 1 in Table 1).*

| *Architecture* | *Pattern 1* | | *Pattern 2* | |
|---|---|---|---|---|
| | $a$ | $b$ | $a$ | $b$ |
| Ethernet/FDDI | $2a_w + a_c$ | $\max(b_w, b_c)$ | $4(n-1)a_w + n(n-1)a_c$ | $\max(2(n-1)b_w, n(n-1)b_c)$ |
| IBM SP2 | $2a_p + a_c$ | $\max(b_p, b_l)$ | $2a_p + na_c$ | $\max(b_p, nb_c)$ |
| Paragon | $2a_p + 2a_c$ | $\max(b_p, b_c)$ | $2a_p + 2na_c$ | $\max(b_p, nb_c)$ |

**Table 1.** *Communication parameters for the equivalent CBs in Figs. 2 and 3.*

As an alternative, we propose a simple experimental approach to determine the communication parameters. For each architecture we consider two communication patterns. In the first, we measure the communication times between two

[3] This is because of various factors like interrupt processing, context switching, memory management etc., and other hardware features like the presence of a cache memory system.

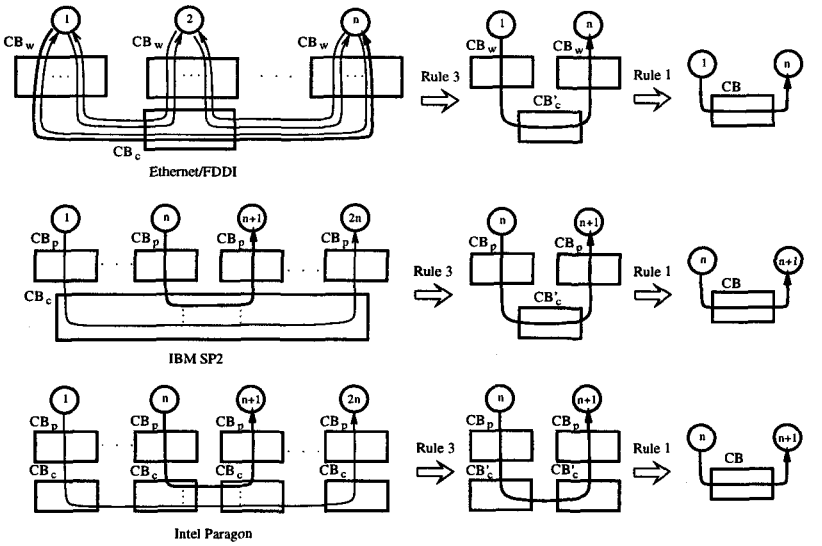


**Fig. 3.** *Communication graphs and their reduction to the equivalent CBs for the all-to-all communication pattern (Ethernet/FDDI), and for the pattern in which each processor i ($n \leq i$) sends a message to the processor $2n-i+1$ (SP2/Paragon) (Pattern 2 in Table 1).*

end-nodes for messages of various sizes. Figure 2 shows the corresponding communication graphs and their reduction (by using Rule 1 for independent blocks) to the equivalent $CB$s for Ethernet NOW, FDDI NOW, IBM SP2, and Intel Paragon. In the Ethernet/FDDI case $CB_c$ models the communication network, while the workstations are modeled by the same $CB_w$ irrespective of whether a message is sent or received. In the IBM SP2 case $CB_p$ models a processor, $CB_c$ models the communication switch, and $CB_l$ models a communication link between two end-nodes.[4] Finally, in the Paragon case $CB_c$ models the dedicated communication processor that resides on each node.

In the second communication pattern we try to send as many messages as possible in order to achieve a communication bottleneck in the network and "expose" the $CB_c$ parameters (that otherwise are "shadowed" by the $CB_w/CB_p$ parameters). For Ethernet/FDDI we use an all-to-all communication pattern, in which every workstation sends/receives a message to/from all others. We were not able to use the same pattern for SP2, since for large messages the program hang. (We suspect buffer overflow within the communication switch is the cause.) We have therefore replaced it with a pattern in which each node sends/receives

[4] Each logical node (processor) of an SP2 belongs to a logical frame, which is a two-stage switch that provides any permutation between 16 bidirectional links to/from 16 processors. In this case, the $CB_c$ parameters are dependent on the interconnection topology. For example, as long as the interconnection topology provides separate paths for all pairs of end-nodes exchanging messages, the communication link between any two end-nodes is modeled simply as $CB_l$.

a message to/from exactly one other node. The same communication pattern is used for the Paragon. Figure 3 depicts communication graphs for each case as well as their reductions (by using Rules 3 and 1) to the equivalent $CB$s. Table 1 shows the communication parameters of the equivalent $CB$s for each case.

| *Architecture* | $a_w$ or $a_p$ | $b_w$ or $b_p$ | $a_c$ | $b_c$ or $b_l$ |
|---|---|---|---|---|
| Ethernet | $750\mu$sec | $1.05\mu$sec/byte | $250\mu$sec | $0.95\mu$sec/byte |
| FDDI | $380\mu$sec | $0.13\mu$sec/byte | $5\mu$sec | $0.11\mu$sec/byte |
| IBM SP2 | $62\mu\text{sec} - a_c/2$ | $0.025\mu$sec/byte | $124\mu\text{sec} - 2a_p$ | $0.0285\mu$sec/byte |
| Paragon | $120\mu\text{sec} - a_c$ | $0.031\mu$sec/byte | $120\mu\text{sec} - a_p$ | $0.012\mu$sec/byte |

**Table 2.** *Communication parameters for the CBs in Figs. 2 and 3.*

Next, by measuring the communication times for small and very large messages (0.5 MB) for each communication pattern, we determine the parameters $a$ and $b$ of the equivalent $CB$s shown in Figs. 2 and 3 (recall from Eq. (2) that $\lim_{x\to 0} T(x,a,b) = a$, and $\lim_{x\to\infty} T(x,a,b) = b$). Finally, from the formulae in Table 1 we can determine the values of the communication parameters for different blocks (see Table 2). In two cases (SP2 and Paragon) we are not able to independently compute $a_c$ and $a_p$, since we can not achieve a communication bottleneck for very short messages. However, this is not a problem in practice since it is unlikely that an application will separately expose these parameters.

## 4 Test Application

A model parallel scientific application [2], chosen because of the high scaling of its communication requirements relative to computation requirements, was rewritten in MPI and instrumented for use as a test program for the hyperbolic model. To verify its accuracy, we select for graphical comparison estimates of the communication times and corresponding measurements. The estimates derive from the archetypal communication operations as described in [4], with parameters evaluated for each platform as in Section 3.

The model application is transient simulation of Navier-Stokes flow in a square fluid-filled cavity, driven by an oscillating rigid lid. Space parallelism is achieved through domain partitioning, with one processor per subdomain. Time parallelism is achieved by assigning identically spatially decomposed time planes to disjoint sets of processors. The motivation for time parallelism is the degradation of efficiency in space parallelism that is due both to increasing perimeter-to-area (surface-to-volume) ratio and to slowing of convergence as spatial coupling is sacrificed. In the limit of pure time parallelism, $p$ processors work concurrently on $p$ different time planes of the transient solution. In the limit of pure space parallelism, only one time plane is computed at a time. The communication patterns and the amount of traffic vary with the allocation of available processors between space and time, as well as with the refinement of the spatial grid. However, for time parallelism, the asymptotic computation and communication

complexity are of equal order, since full planes of data must be transmitted between time levels for every plane-based computational update.

We derive overall communication parameters for the traffic consisting of inter-plane grid transfers, which are dominant in overall communication complexity. On a homogeneous set of processors this pattern should exhibit contention since all transfers will start at "almost" the same time. Figure 4 shows the communication graphs corresponding to this communication pattern for Ethernet/FDDI, IBM SP2, and Intel Paragon, as well as the reduction to an equivalent $CB$. In the case of the SP2 we assume that every pair of nodes communicates through a "dedicated" link (represented by $CB_l$). Table 3 shows the expressions of the communication parameters for the intermediate communication graph, as well as for the final $CB$. These parameters are computed for the bi-directional case where neither nodes $i$ nor $i+1$ are associated with the first or last planes (i.e., $i = 2, 3, \ldots, n-2$). For the first and last planes, communication is uni-directional only and the parameters can be computed in the same manner.

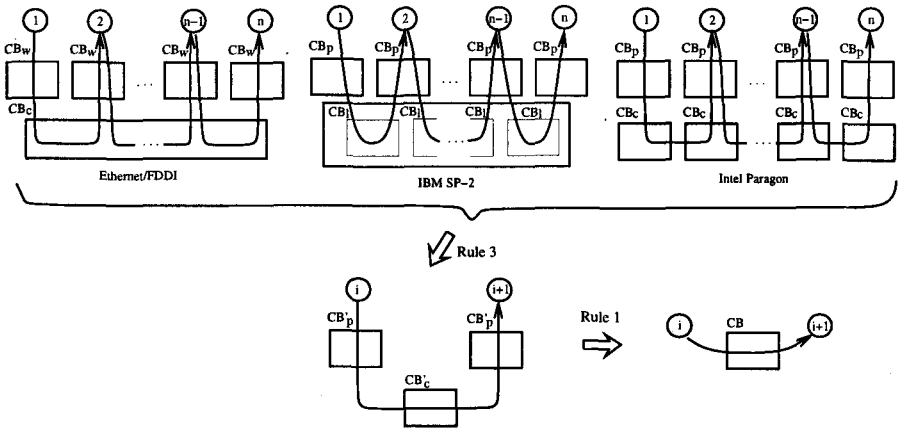


**Fig. 4.** *The communication graph and its reduction to an equivalent CB for the main communication pattern induced by the time-parallel application in the time-parallel limit for: Ethernet/FDDI, IBM SP2 and Intel Paragon*

| *Architecture* | $CB'_p$ | | $CB'_c$ | | $CB$ | |
|---|---|---|---|---|---|---|
| | $a'_p$ | $b'_p$ | $a'_c$ | $b'_c$ | $a$ | $b$ |
| Ethernet/FDDI | $2\ a_w$ | $2\ b_w$ | $(n-1)a_c$ | $(n-1)b_c$ | $4a_w + (n-1)a_c$ | $\max\{2b_w, (n-1)b_c\}$ |
| IBM SP2 | $2\ a_p$ | $2\ b_p$ | $a_l$ | $b_l$ | $4\ a_p + a_l$ | $\max\{2b_p, b_l\}$ |
| Intel Paragon | $2\ a_p$ | $2\ b_p$ | $2\ a_c$ | $2\ b_c$ | $4\ a_p + 2\ a_c$ | $\max\{2b_p, 2b_c\}$ |

**Table 3.** *Communication parameters for the CBs shown in Fig. 6.*

The application has been ported under MPI to the four platforms evaluated in Section 3. For Ethernet and FDDI we run the experiments on up to 8 SUN

SPARCstation 20s. For the SP2 and Paragon, in choosing the maximum numbers of nodes, our goal is to minimize as much as possible the interference of other users. Therefore, on the SP2 we run the experiments by using up to the maximum number of processors on a frame (i.e., 16), and for the Paragon we run the experiments on up to 12 nodes, since this is the maximum number of nodes we can allocate along a mesh column. (By allocating all the nodes on one side of the mesh we eliminate any potential interference.)

Figure 5 shows the predicted versus the measured values for the total communication times corresponding to one iteration in the algorithm. For FDDI, the SP2, and the Paragon the predicted data are within 20% of the measurements.[5]

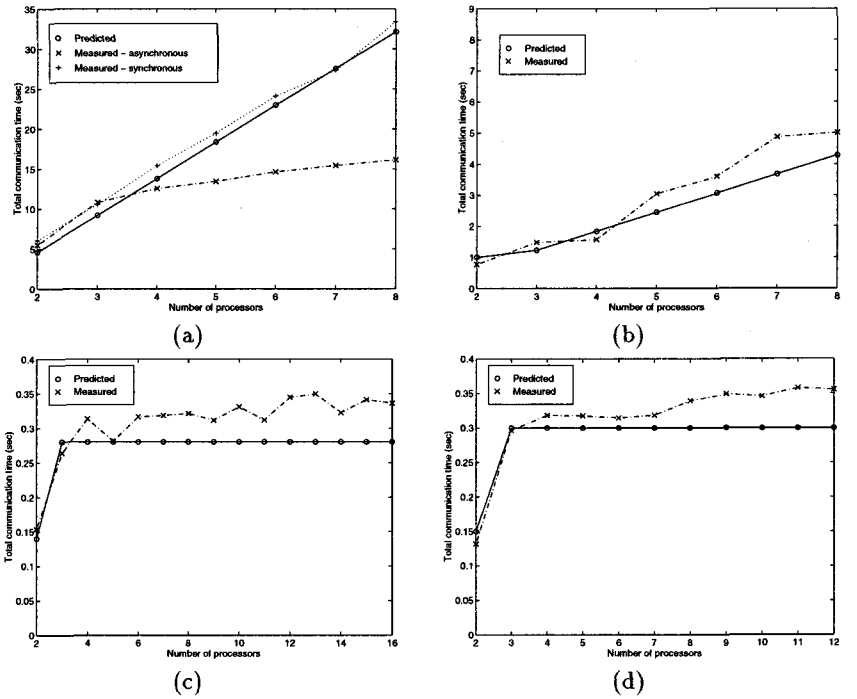


**Fig. 5.** *Predicted versus measured communication times for: (a) Ethernet NOW, (b) FDDI NOW, (c) IBM SP2, (d) Intel Paragon.*

On the other hand, for Ethernet the difference between the predicted data and the measurements (the "asynchronous" curve) is much larger and tends to increase with the number of processors. The main reason is that the hyperbolic model assumes that all processors send data at the same time, which yields an upper bound on the communication time. Though the application is inherently

[5] The exception is the seven-processor experiment under FDDI, which we believe was primarily due to communication interference from other workstations on the same subnet. (We reserved only the individual workstations; we could not reserve the entire subnet.)

synchronous, in practice the probabilistic protocol employed by Ethernet "destroys" the synchronicity. At the beginning of each iteration all workstations attempt to send messages "almost" at the same time, and therefore the probability of collision is high. When a workstation detects such a collision, it waits a random amount of time [3], before retransmitting the packet. In time, this results in workstations sending out paackets at slightly staggered intervals. Consequently, the degree of overlap between messages sent by different processors is much lower than is assumed in the model. For validation purposes, we changed the algorithm to force synchronization at intermediate points during an iteration. As shown in Fig. 5, the measured overall communication time in this case (the "synchronous" curve) is very close to the predicted value.

The key contrast between the NOWs and the tightly-coupled machines, as predicted by the model and as borne out in the experiments, is in the asymptotic behavior of the communication time with respect to the number of processors. For the NOWs it is linear, since the communication network is a shared resource of limited capacity: adding more nodes decreases the share of communication bandwidth allocated to each processor.

For the SP2 and the Paragon, the communication times remain practically constant as the number of nodes increases. This is expected from the scalability of the communication subsystems employed by these platforms. The difference in the time to complete the communication in going from two to three processors for the SP2 and the Paragon is due to the presence, for more than two processors, of at least one processor (in between) which both sends and receives data. In the two-processor case each processor either sends or receives (but not both), which reduces by nearly half the overall message processing time at the end nodes.

In conclusion, the two-parameter hyperbolic model [4] for parallel communication complexity on general dedicated networks has been applied, using a uniform set of rules, to a variety of architectures employing a variety of message topologies. The model is flexible and reasonably accurate in predicting the communication times for an archetypal application.

## References


1. Hockney, R. W. Performance Parameters and Benchmarking of Supercomputers. *Parallel Computing*, **17** (1991), 1111-1130.
2. Horton, G. Time-parallelism for the massively parallel solution of parabolic PDEs, in *Applications of High Performance Computers in Science and Engineering*, Computational Mechanics Publications, Southampton (UK), December 1994.
3. Stallings, W. *Data and Computer Communications*, Macmillan, New York, 1991.
4. Stoica, I., Sultan, F., Keyes, D. A Simple Hyperbolic Model for Communication in Parallel Processing Environments, *ICASE TR 94-78*, September 1994. To appear in *Journal of Parallel and Distributed Computing.*
5. Valiant, L. G. A Bridging Model for Parallel Computation. *Communications of the ACM*, **33** (1990), 103-111.