C^3: An architecture-independent model for coarse-grained parallel machines[*]

Susanne E. Hambrusch
Department of CS
Purdue University
West Lafayette, IN 47907
seh@cs.purdue.edu

Ashfaq A. Khokhar
School of EE and Dept. of CS
Purdue University
West Lafayette, IN 47907
ashfaq@cs.purdue.edu

Abstract

We propose an architecture-independent parallel model, the C^3-model. The C^3-model evaluates, for a given parallel algorithm and target architecture, the complexity of computation, *the pattern of* communication, *and the potential* congestion *arising in communication operations. A metric for estimating the effect of link and processor congestion on the performance of an arbitrary communication operation is developed. We describe how the C^3-model can serve as a platform for the development of coarse-grained algorithms sensitive to the parameters of a parallel machine. The initial validation of the C^3-model is discussed through different implementations of communication operations on the Intel Touchstone Delta.*

1 Introduction

The development of a parallel model that bridges the gap between software and hardware has been recognized crucial to the success of massively parallel computation. Such a model should be simple, should accurately reflect the constraints of a parallel machine, and should have broad applicability with respect to existing machines. In addition, such a model should provide a platform for algorithm development and allow accurate prediction of the performance of an algorithm. Recently, a number of models with this goal have been proposed [3, 5, 8, 10, 11, 15, 16, 17]. In most of these models, processors are assumed to communicate using a point-to-point message router. Composing more involved communication operations by having to specify fine-scheduling details places a significant burden on application programmers. Furthermore, the above models do not attempt to capture the effect of link or processor congestion.

*Research supported in part by ARPA under contract DABT63-92-C-0022ONR. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing official policies, expressed or implied, of the U.S. government.

In this paper, we propose an architecture-independent parallel model, the C^3-model. This model captures the complexity of *computation*, the pattern of *communication*, and the potential *congestion* during communication. We provide a metric for estimating the effect of link and processor congestion on the performance of communication operations. Our metric allows the evaluation of arbitrary communication operations without having to specify fine scheduling details. We investigate how well the C^3-model serves as a platform for the development of coarse-grained algorithms and as a tool for estimating the performance of an algorithm. For the initial validation of the C^3-model, communication operations have been implemented on the Intel Touchstone Delta, and performance results are discussed and compared to the predicted performance.

We assume that computation is synchronized by a barrier-style synchronization mechanism similar to the one described in [16]. More precisely, an algorithm can be partitioned into a sequence of supersteps, with each superstep corresponding to local computation followed by sending and receiving messages. Synchronization occurs between supersteps. We express the performance of a superstep, and thus of an algorithm, in terms of *computation units* and *communication units*. Counting in units allows us to penalize certain undesirable aspects in local computations and in communication operations. The number of computation units charged depends on the amount of local computation done. The number of communication units charged depends on the amount of data sent and the amount of data received by a processor, the latency encountered by the messages, the congestion arising due to the volume of inter-processor communication, and the routing schema and routing protocols used. Our method for evaluating communication units estimates the effect of these factors on the performance.

Section 2 describes the C^3-model and the metric used to determine communication and computation units. In Section 3, we use straightforward implementations of common communication operations as examples for our metric. In Section 4, we use the C^3-

model to develop and analyze a number of different implementations of these operation and we compare the predicted performance to the results achieved on the Intel Delta.

2 The C^3-Model

We first state machine parameters used for determining computation and communication units. Let p be the number of processors available in the machine. In current coarse-grained machines, the computing power of a processor is equivalent to that of a state-of-the-art workstation. We assume that algorithms on coarse-grained machines are not constrained by the amount of local memory. A message is made up of fixed-length packets and a packet is the logical unit for communication between two processors. We use l to denote the length of a packet (measured in bytes), s to denote the set-up cost for a message, and h to denote the latency. We use the average distance between two processors to represent the latency. In the subsequent discussion we assume a processor bandwidth and a network bandwidth of l. How to include different bandwidth parameters into our charging method is described later.

Numerous applications contain routing steps in which the p processors are partitioned into q sets $S_1, \ldots S_q$, with S_i containing p_i processors, such that sends and receives are issued only between processors in the same set. Assume every processor set S_i corresponds to a scaled down version of size p_i of the p-processor machine. If hardware and software support the execution of operations within processor set S_i, independent of the operations done in the other submachines, efficiency of routing operations is enhanced significantly. The importance of being able to operate on independent submachines has been recognized. It has been incorporated into the Message Passing Interface (MPI) [6] and has been extended to arbitrary process groups [1]. In our evaluation of communication units we assume that independent routing in submachines is possible. We thus charge communication units based on the parameters of the associated submachines.

2.1 Computation Units

The charging of computation units in a superstep is done as follows. Assume that in one superstep processor P_i accesses t_i bytes. The superstep is charged $\max_{0 \le i \le p-1} \lceil \frac{t_i}{l} \rceil$ computation units. The reason for normalizing computation units by l is that too little computation between two communication steps should have a negative impact on the performance. If $t_i < l$, we charge one computation unit and thus also penalize for not accessing enough bytes to fill a packet.

2.2 Communication Units

The communication units charged to one superstep reflect the time spent in sending and receiving messages, the time messages are en-route under ideal conditions, the amount of congestion that could occur, and an estimate on the resulting delay. The number of communication units charged also depend on the type of routing schema and the type of send and receive primitives used. The two routing schemas we consider are ***store-and-forward*** and ***wormhole routing***. Most existing machines support both blocking and nonblocking protocols for send and receive primitives. These protocols differ in implementation based on the synchronization methods used. In this paper we consider nonblocking receives together with blocking and nonblocking sends. For clarity, a ***blocking send*** is a send operation initiated by a source processor which does not terminate until the message is received by the destination processor. In a ***nonblocking send*** the source processor, after filling its send buffer, has to wait only until the message has been read out of the send buffer.

Assume processor P_i sends a message consisting of $L_{i,j}$ bytes to processor P_j, $0 \le i,j \le p-1$. This message uses a ***send time*** $s_{i,j}$, which is an estimate on the time needed to send the message when it encounters no congestion. For nonblocking sends and nonblocking receives, it is $s + \lceil \frac{L_{i,j}}{l} \rceil \star h$ for store-and-forward and $s + \lceil \frac{L_{i,j}}{l} \rceil + h$ for wormhole routing. For blocking sends and nonblocking receives, the send time is $2(s+h) + \lceil \frac{L_{i,j}}{l} \rceil \star h$ for store-and-forward and $2(s+h) + \lceil \frac{L_{i,j}}{l} \rceil + h$ for wormhole routing. Processor P_j experiences a ***receive time*** $r_{i,j}$ which represents the time P_j is occupied with receiving the message. It is $\lceil \frac{L_{i,j}}{l} \rceil$ for store-and-forward and wormhole using nonblocking sends and receives. For blocking sends and nonblocking receives, it is $s + h + \lceil \frac{L_{i,j}}{l} \rceil \star h$ for store-and-forward and $s + 2h + \lceil \frac{L_{i,j}}{l} \rceil$ for wormhole routing. Let $n_s(i)$ and $n_r(i)$ denote the number of processors to which P_i sends a message and from which it receives a message, respectively. The ***total send*** and ***total receive*** times, S_i and R_i, for processor P_i in a superstep represent bounds on sending and receiving these messages in a congestion-free environment. Figure 1 gives the total send and receive times experienced under different routing protocols.

We briefly comment on the quantities given for store-and-forward routing with nonblocking sends and nonblocking receives. Let P_j' be the first processor to whom P_i issues a send. After $s + \lceil \frac{L_{i,j'}}{l} \rceil$ steps, processor P_i is no longer engaged in the send process and can proceed with the next send, thus pipelining the $n_s(i)$ sends. The total send time S_i includes $n_s(i)$ message set-up costs, the total number of packets sent out, and the latency experienced by sending out the last packet

Protocol*	S_i	R_i
SF, nbs, nbr	$s \star n_s(i) + h \star \lceil \frac{L_{i,jmax}}{l} \rceil + \sum_{0 \leq j \leq p-1} \lceil \frac{L_{i,j}}{l} \rceil$	$\sum_{0 \leq j \leq p-1} \lceil \frac{L_{j,i}}{l} \rceil$
WH, nbs, nbr	$s \star n_s(i) + h + \sum_{0 \leq j \leq p-1} \lceil \frac{L_{i,j}}{l} \rceil$	$\sum_{0 \leq j \leq p-1} \lceil \frac{L_{j,i}}{l} \rceil$
SF, bs, nbr	$2(s+h) \star n_s(i) + h \star \sum_{0 \leq j \leq p-1} \lceil \frac{L_{i,j}}{l} \rceil$	$(s+h) \star n_r(i) + h \star \sum_{0 \leq j \leq p-1} \lceil \frac{L_{j,i}}{l} \rceil$
WH, bs, nbr	$2(s+h) \star n_s(i) + h + \sum_{0 \leq j \leq p-1} \lceil \frac{L_{i,j}}{l} \rceil$	$(s+h) \star n_r(i) + h + \sum_{0 \leq j \leq p-1} \lceil \frac{L_{j,i}}{l} \rceil$

Figure 1: Total send and receive times for processor P_i under different routing protocols.
*SF = Store and Forward, WH = wormhole routing, nbs = nonblocking sends, nbr = nonblocking receives, bs blocking sends, br = blocking receives

(which is bounded by the largest message size).

The quantity $S_i + R_i$ represents a bound on the time processor P_i spends in one superstep on sending and receiving messages. Charging one superstep $\max_{0 \leq i \leq p-1}\{S_i + R_i\}$ communication units reflects the overall send and receive time experienced by the machine during the communication operation, not including the delay the messages encounter because of link and processor congestion. We point out that when stating communication units, so far we have not scaled the set-up cost but simply have included the total number of set-up costs experienced. When giving communication units for operations on specific machines in Section 4, we convert set-up costs to communication units.

Congestion plays a crucial role in achieving high performance. At the same time, congestion is difficult to evaluate. Congestion is a global phenomena and where it occurs depends on specifics of the architecture and the routing paths taken. A formal model to deal with contention in a shared memory machine has been proposed in [7]. In general, congestion depends on the amount of data sent between processor pairs. The amount is independent of whether we use store-and-forward or wormhole routing. In our estimation of congestion, we measure C_l, the congestion over links, and C_p, the congestion at the processors. We measure processor and link congestion under the assumption that all messages are routed simultaneously. Clearly, this may not be done under a given protocol. However, delaying the sending of a message by using blocking sends is, in some sense, a possible way of dealing with the congestion. In both cases, the messages experience a delay. The parameters used to measure potential congestion are the following:

- p, the number of processors,
- $cong$, the total number of processor pairs communicating,
- b, the bisection width of the machine, and
- L_a, be the average number of packets routed between the processors.

Congestion over links is closely related to the bisection width of the machine. In a machine with a bisection width of b, it takes at least $\lceil \frac{K}{b} \rceil$ steps to send K packets from processors in one half of the machine to the processors in the other half. We set

$$C_l = L_a \star \lceil \frac{cong}{b} \rceil.$$

Our estimation of the link congestion C_l is both optimistic and pessimistic. It is optimistic in measuring congestion only over a single link cut (namely, the cut that separates the machine into halves). It is pessimistic in assuming that all *cong* communicating processor pairs have the source processor in one half and the destination processor in the other half of the machine.

In order to estimate the congestion at the processors assume that all *cong* processor pairs are routed simultaneously. Processor congestion is then estimated as

$$C_p = L_a \star \lceil \frac{cong}{p} \rceil \star h.$$

The quantity $\lceil \frac{cong}{p} \rceil$ represents the average number of messages at a processor at the beginning of the communication operation. We use L_a, the average message length, in estimating the slow-down a message experiences. We argue that a message of size L_a traversing a distance of h links and thus competing for the resources with other messages at each of the $h-1$ intermediate processors is slowed down by a factor of $\lceil \frac{cong}{p} \rceil$ at each processor. We do not take into account that congestion at the processors is likely to decrease during the routing. Capturing this behavior in a simple way is difficult and in many realistic routings (e.g., a transpose) the decrease in the congestion is slow.

In summary, the total number of communication units charged in a superstep is

$$\max_{0 \le i \le p-1} \{S_i + R_i\} + C_l + C_p.$$

In order to estimate actual execution time of an algorithm, relative weights need to be attached to computation and communication units. These weights should be based on the ratio between the processor clock speed and the network clock speed as well as the ratio of the bandwidth of the network and the bandwidth of the processors. In the high-level approach taken by our model, clock speeds and bandwidth parameters do not influence the design of an algorithm and they are thus not included. Put in a different way, we give units for the case when the network clock speed is equal to processor clock speed and network bandwidth is equal to processor bandwidth. When evaluating an algorithm the ratio of computation units and communication units over all supersteps gives information as to whether an algorithm is computation or communication intensive.

3 Charging Communication Units

In this section we use different communication patterns to demonstrate our method for charging communication units in a superstep. Our metric allows the evaluation of arbitrary communication patterns. While arbitrary patterns occur in applications, regular patterns are more common on coarse-grained machines. We give the number of communication units charged for regular patterns when each communication operation is implemented using the naive approach of each processor sending messages directly to the destination processors. The communication operations we consider include one-to-one, one-to-all, all-to-one, and all-to-all routing. The communication units are given for wormhole routing with nonblocking sends and nonblocking receives. To simplify the presentation, we assume that every message is of length L.

In *one-to-one* routing, also known as permutation routing, every processor sends L bytes to a unique destination (i.e., unique among all p processors). Our charging method does not distinguish between routings that are easy or difficult with respect to the arising congestion. Clearly, for any particular architecture, such differences do exist. In one-to-one routing we have $n_s(i) = 1$, $n_r(i) = 1$, $0 \le i \le p-1$, and $cong = p$. Figure 2 gives total send and total receive times, link and processor congestion for one-to-one and other communication operations.

For one-to-one routing, link and processor congestion dominate the communication units. Whether one can expect more congestion over the links or at the processors, depends on the bisection width of the machine. Assume that one-to-one routing is done on a p-processor square mesh with $b = \sqrt{p}$ and $h = \frac{2}{3}\sqrt{p}$. Then, processor and link congestion appear almost balanced and we charge

$$s + \frac{2}{3}\sqrt{p} + \lceil \frac{L}{l} \rceil \star (2 + \frac{5}{3}\sqrt{p})$$

communication units. On a p-processor hypercube we have $b = p/2$ and $h = \frac{\log p}{2}$ and the processor congestion dominates. In total, we charge

$$s + \frac{\log p}{2} + \lceil \frac{L}{l} \rceil \star (4 + \frac{\log p}{2})$$

communication units. On a tree machine with $h = \log p$ and $b = 1$ link congestion dominates and we charge

$$s + \log p + \lceil \frac{L}{l} \rceil \star (2 + p + \log p).$$

In *one-to-all* routing, one processor, say processor P_t, sends $p-1$ distinct messages, each to a different destination. We have $n_s(t) = p-1$, $n_r(i) = 1$ for $i \ne t$, $0 \le i \le p-1$, and $cong = p-1$. Clearly, the total send time experienced by the source processor P_t dominates the number of communication units.

All-to-one routing is the inverse of one-to-all: every processor now sends a message to a common processor, say processor P_t. We have $n_s(i) = 1$ for $i \ne t$, $0 \le i \le p-1$, $n_r(t) = p-1$, and $cong = p-1$. The total receive time at processor P_t dominates the number of communication units.

In *all-to-all* routing, also known as total exchange, every processor sends a message to every other processor. We have $n_s(i) = p-1$, $n_r(i) = p-1$, $0 \le i \le p-1$, and $cong = p(p-1)$. From the number of communication units charged shown in Figure 2 it follows that link and processor congestion dominate the number of communication units. In the next section we use the C^3 model to develop a family of algorithms for each one of the communication operations.

4 C^3 as a Platform for Developing Communication Operations

Efficient communication operations are crucial for making programs scalable and portable across different machines. Common communication operations should be implemented with the specific features and parameters of the machine in mind. Implementing operations through independent sends and receives is not likely to result in the best implementation. In this section we use the C^3-model as a platform to develop and analyze different implementations of communication operations. For each implementation we compute computation and communication units and compare total units to the performance of the algorithms on the Intel Delta. The Delta uses wormhole

	S_i	R_i	C_l	C_p
one-to-one	$s + \lceil \frac{L}{l} \rceil + h$	$\lceil \frac{L}{l} \rceil$	$\lceil \frac{L}{l} \rceil \star \lceil \frac{p}{b} \rceil$	$\lceil \frac{L}{l} \rceil \star h$
one-to-all	$(p-1) \star (s + \lceil \frac{L}{l} \rceil) + h,\ i = t$	$\lceil \frac{L}{l} \rceil,\ i \neq t$	$\lceil \frac{L}{l} \rceil \lceil \frac{p-1}{b} \rceil$	$\lceil \frac{L}{l} \rceil \star h$
all-to-one	$s + \lceil \frac{L}{l} \rceil + h,\ i \neq t$	$\lceil \frac{L}{l} \rceil \star (p-1),\ i = t$	$\lceil \frac{L}{l} \rceil \lceil \frac{p-1}{b} \rceil$	$\lceil \frac{L}{l} \rceil \star h$
all-to-all	$(p-1) \star (s + \lceil \frac{L}{l} \rceil) + h$	$\lceil \frac{L}{l} \rceil \star (p-1)$	$\lceil \frac{L}{l} \rceil \lceil \frac{p(p-1)}{b} \rceil$	$\lceil \frac{L}{l} \rceil \star h \star (p-1)$

Figure 2: Communication units charged for wormhole routing with nonblocking sends and nonblocking receives

routing and allows the use of blocking as well as nonblocking sends. We give communication units and performance for wormhole routing with nonblocking sends and nonblocking receives. Our results indicate that the efficiency of a communication operation is influenced by the relationship among parameters of the parallel machine, as well as by the relationship of the parameters to the amount of data involved. This agrees with other research done on the implementation of communication operations, [1, 2, 4, 12].

In order to classify the different approaches used in our implementations, we introduce the notion of a k-level algorithm. Intuitively, in a k-level algorithm the machine is partitioned into k levels of submachines, with the submachines within each level operating independently from each other. An algorithm is a *1-level* algorithm if, in the description given in terms of supersteps, no superstep operates on independent submachines. In a *k-level* algorithm, $k > 1$, there exists at least one superstep that assumes a partition of the machine into independent submachines and the following supersteps specify a $(k-1)$-level algorithm for each submachine.

When describing our algorithms, we assume, for the sake of simplicity, that the size of the message routed between any two processors is L. We refer to L as the *actual message size*. Our k-level algorithms are characterized by combining the original messages of size L and by performing routings within independent submachines. Therefore, in a superstep the size of the message routed between two processors can be different from the actual message size. We refer to the size of a message routed between processors as the *effective message size*. For all algorithms, the effective message size is never smaller than the actual message size.

4.1 One-to-all Routing

In this section, we use the k-level concept to develop a number of different implementations for one-to-all routing. We make a number of simplifications when giving communication units. First, we write p when the correct quantity is $p-1$. We also may omit additive terms of h.

There exist two conceptually different 1-level one-to-all algorithms. In the first one, Algorithm *1-lev-dir*, source processor P_t issues $p-1$ direct sends. Using Figure 2, our model charges $sp + \lceil \frac{L}{l} \rceil \star (p + \lceil \frac{p}{b} \rceil + h)$ communication units.

Another 1-level approach is to have processor P_t form one long message of size $L(p-1)$ which is broadcast to every processor and then let each processor extract its message from the long message received. One expects the broadcasting approach to be efficient only when L is small and when the parallel machine has a control network dedicated to fast broadcasts. We considered two versions of this approach. The first one, Algorithm *1-lev-sys-br*, uses the system's broadcast and the second one, Algorithm *1-lev-our-br*, uses a binomial heap as a broadcasting tree.

We next describe a generic 2-level approach. Logically partition the p-processor machine into p^{α} submachines, each containing $p^{1-\alpha}$ processors for $\frac{1}{\log p} \leq \alpha < 1$. Designate one processor in each submachine as a leader. Source processor P_s then forms p^{α} long messages, each having an effective message size of $Lp^{1-\alpha}$. The i-th long message formed consists of the $p^{1-\alpha}$ actual messages destined for the processors in the i-th submachine, $0 \leq i < p^{\alpha} - 1$. Next, processor P_s issues p^{α} sends (or $p^{\alpha} - 1$ sends if P_s is a leader) to route the long messages to the leaders. Once a leader has received its long message, it divides the message into $p^{1-\alpha}$ of size L and initiates a 1-level one-to-all algorithm within its submachine. For the mesh architecture, we considered such a 2-level algorithm, Algorithm *2-lev-rec*, in which each submachine consists of a row of processors. In this case, the first superstep operates on a single column of the mesh. The second superstep uses Algorithm *1-lev-dir* within each row.

The number of communication units charged is

$$s\sqrt{p}+\lceil \frac{L\sqrt{p}}{l} \rceil \star (\sqrt{p}+\lceil \frac{\sqrt{p}}{b'} \rceil + h') + s\sqrt{p}+\lceil \frac{L}{l} \rceil \star (\sqrt{p}+\lceil \frac{\sqrt{p}}{b'} \rceil + h'),$$

where b' and h' are the bisection width and the average distance in a $\sqrt{p}$-processor linear array, respectively.

A 3-level algorithm is obtained by applying a 2-level approach to submachines. We have implemented the following 3-level algorithm, Algorithm *3-lev-sq*. The p-processor machine is logically partitioned into $\sqrt{p}$ submachines, each being an array of size $p^{1/4} \times p^{1/4}$. Once a leader receives its long message from P_t, it initiates a 2-level algorithm for one-to-all routing (using Algorithm *2-lev-rec*) within its submachine.

The value of $k = \log p$ leads to a class of interesting algorithms to which we refer as Binomial Heap algorithms. A p-processor machine is now divided into two submachines and the source processor P_t issues one send to the leader in the other submachine. After this send, a $(k-1)$-level algorithm is invoked and it proceeds in the same fashion. When the machine is divided into submachines of equal size, we perform $\log p$ superstep minimizing the total number of set-up costs.

We have implemented a number of algorithms based on the binomial heap approach on the Delta. Algorithm *logp-lev-sq* divides the mesh into half by alternating vertical and horizontal cuts. Let $CBH(p)$ be the number of communication units charged to Algorithm *logp-lev-sq* on a p-processor machine. Then, $CBH(p) \leq s \log p + c_1 \star \lceil \frac{Lp}{l} \rceil \star h$, for a constant $c_1 \leq 1.5$. Another $\log p$-level approach is to divide the p-processor machine into two submachines of uneven size. For a given value γ, $0 < \gamma < 1$, we thus form one submachine containing the source processor and a total of γp processors, $0 < \gamma < 1$, while the other submachine contains the remaining processors. The communication units for an algorithm based on this approach are derived in a similar fashion, using slightly different values for the latency and the bisection width.

The total number of communication and computation units charged in the C^3-model to each of the above described algorithms, assuming nonblocking sends and nonblocking receives, are shown in Figures 3. When converting the set-up cost s to units, we assume $s = 1400$ processor cycles. Assuming 40MHz processor clock speed and 12.5 MB/sec network bandwidth, the number of units corresponding to one set-up cost is approximately 8. From the communication units charged, it appears that Algorithm *3-lev-sq* is the best for message sizes of upto 6Kbytes, and Algorithm *1-lev-dir* performs better for larger messages sizes.

We have implemented the above described algorithms on the Intel Delta. We considered machine sizes from 16 to 256 processors and message sizes from 16 bytes to 16Kbytes. The experimental results for $p = 256$ are shown in Figure 5. Expressing each implementation in terms of communication and computation units gives an accurate prediction of the relative performance between different one-to-all algorithms. As indicated by the units, Algorithm *1-lev-dir* is a reasonable choice only for large message sizes. It minimizes the effective message size, but experiences a total of $p-1$ message set-up costs. The two broadcasting algorithms give the worst performance of all algorithm. The poor performance is partly due to the large effective message size, as well as due to the absence of a dedicated fast broadcasting network. Algorithms *2-lev-rec* and *3-lev-sq* perform consistently better than all the other algorithms. The algorithm that gives optimal or near optimal results for all machine and message sizes on Delta is a Binomial heap algorithm with $\gamma = 0.75$, Algorithm *logp-lev-rec*(0.75). The value $\gamma = 0.75$ captures characteristics of the send and receive ratio of the Delta that our model does not attempt to evaluate. Our metric evaluates this algorithm no better than logp-lev-sq.

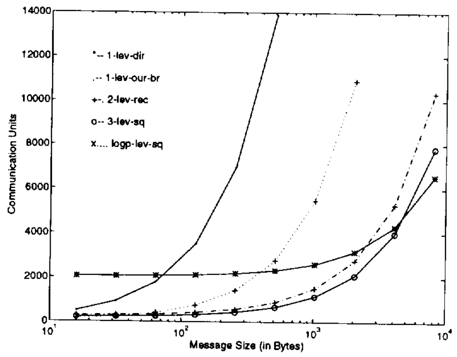

Figure 4: Predicted performance (in units) of the One-to-All Algorithms on a 256-Processor Intel Delta using nonblocking sends and nonblocking receives.

In summary, our validation work on the Intel Delta indicates that the message-combining algorithms which keep a balance between the total number of sends and the effective message size perform well for small message sizes. Which one of them gives the best performance depends on the ratio between the send and receive time, the packet length, the ratio between the processor and network bandwidth, and the message set-up cost.

4.2 All-to-one Routing

In all-to-one routing every processor sends a unique message to one common processor. Conceptually, all-to-one routing is the inverse of one-to-all. We have evaluated and implemented four all-to-one algorithms, namely algorithms *1-lev-dir*, *2-lev-rec*, *3-lev-*

Algorithm	Communication Units	Computation Units	Communication Units (with s=8)
1-lev-dir	$256s + 0.55L$	$\frac{L}{512}$	$2048 + 0.55L$
1-lev-our-br	$8s + 27L$	L	$64 + 27L$
2-lev-rec	$32s + 1.23L$	$\frac{L}{32}$	$256 + 1.23L$
3-lev-sq	$24s + 0.93L$	$\frac{L}{25}$	$192 + 0.93L$
logp-lev-sq	$8s + 5.29L$	L	$64 + 5.29L$

Figure 3: Approximate number of units charged for one-to-all algorithms assuming a 256-processor Intel Delta with $h = 10$, $l = 512$, and $b = 16$.

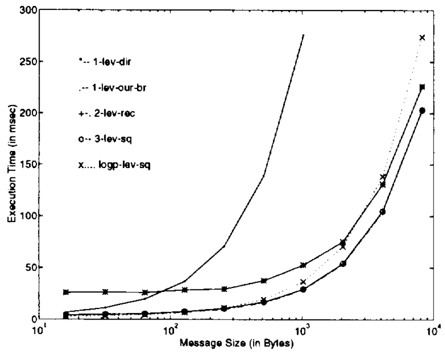

Figure 5: Experimental results of the One-to-All Algorithms on a 256-Processor Intel Delta using nonblocking sends and nonblocking receives.

sq, and *logp-lev-sq*. The number of communication units charged for each of the algorithms is almost identical to the ones charged for corresponding one-to-all algorithm. On the Delta, the best one-to-all algorithms did not correspond to the best all-to-one algorithms. For a complete discussion we refer to [9].

4.3 All-to-all Routing

The most straightforward 1-level approach for all-to-all routing is to have each processor send its $p - 1$ messages, one by one, regardless of what other processors are doing. This approach is used in Algorithm *1-lev-dir*. In this algorithm the machine is flooded with messages and the arising congestion is left to be handled by the system. A frequently used approach that attempts to control congestion is to implement all-to-all through $p - 1$ one-to-one routings; i.e., the $p(p - 1)$ routing requests are partitioned into permutations. Common are linear permutations and exclusive-or permutations. Implementations of these approaches on different machines have shown exclusive-or permutations to be superior to linear permutations [12, 13]. Another interesting approach for partitioning all-to-all routings into permutations has been introduced in [14]. We call this approach partitioning into *balanced* permutations. Balanced permutations are relevant for mesh architectures since they result in a smaller congestion over the links compared to linear and exclusive-or permutations. We refer to an algorithm that performs all-to-all routing by partitioning into permutations as Algorithm *1-lev-perm*. Our metric charges the same number of communication units for algorithms which partition into p permutations and Algorithm *1-lev-dir*. The number of supersteps and the amount of congestion in each superstep for both of these 1-level approaches is different, but the total number of units charged is the same.

We also considered two 2-level algorithms, Algorithm *2-lev-sq* and Algorithm *2-lev-r,c*, and a $\log p$-level algorithm, Algorithm *logp-lev-bfly* which is based on the butterfly communication pattern. The approach used in the Algorithm *2-lev-sq* is independent of the underlying architecture. It consists of 3 steps: In each step of the algorithm every processor sends out a total of pL bytes; the first and the last step send out pL bytes in the form of $\sqrt{p}$ messages and the second step sends them out as one single message. Algorithm *2-lev-r,c* consists of only 2 steps, with each step sending out a total of pL bytes in the form of $\sqrt{p}$ messages. The approach used in this algorithm is tailored towards the mesh architecture. For a detailed description we refer to [9]. We have implemented the above mentioned algorithms on a 256-processor Intel Delta. The implementation results are compared to the predicted performance in Figures 6 and 7.

The experimental results show that Algorithm *2-lev-c,r* performs best for small message sizes (≤ 256 bytes). Algorithm *2-lev-sq* gave the second best performance for small message sizes. The reason *2-lev-c,r* outperformed *2-lev-sq*, lies in the fact that *2-lev-sq* is a 3-step algorithm (which sends out data three times), while *2-lev-c,r* is a 2-step algorithm. The advantage of the 3-step algorithm is that it uses square meshes as submachines, whereas the 2-step one uses linear arrays. Algorithm *1-lev-perm* using exclusive-or performs best for larger message sizes. As the metric proposed in this paper does not distinguish between different 1-level algorithms, the predicted performance for all 1-level algorithms follow the same curve. However, in actual implementations different permutations

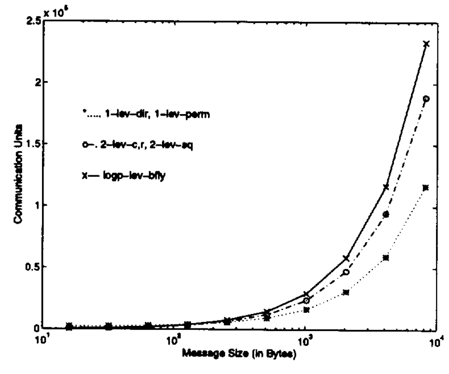

Figure 6: Predicted performance (in units) of the All-to-All Algorithms on a 256-Processor Intel Delta using nonblocking sends and nonblocking receives.

induce different patterns of link and processor congestion and thus give a different performance. Capturing this behavior in the model and its metric would be difficult. The approach in Algorithm ***logp-lev-bfly*** has consistently been judged as being expensive for large message sizes [4, 13]. Our metric and the observed performance on the Delta, confirms that as well.

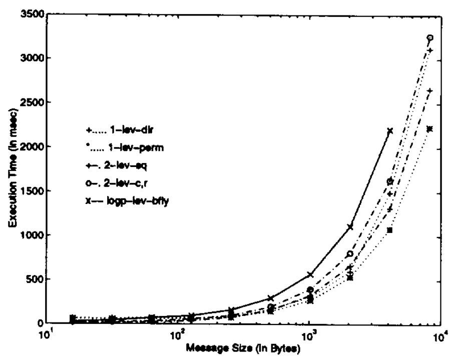

Figure 7: Experimental results of the All-to-All Algorithms on a 256-Processor Intel Delta using nonblocking sends and nonblocking receives.

References

[1] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir, "CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computers," *Proc. of IPPS*, pp. 835-844, 1994.

[2] M. Barnett, R. Littlefield, D. G. Payne, and R. van de Geijn, "Global Combine on Meshes Architecures with Wormhole Routing," *Proc. of IPPS*, pp. 156-162, April 1993.

[3] A. Bar-Noy, S. Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems," *Proc. of SPAA*, pp. 13-22, 1992.

[4] S.H. Bokhari, "Multiphase Complete Exchange on a Circuit Switched Hypercube," *Proc. of ICPP*, pp. 525-529, 1991.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Proc. of 4-th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming*, pp. 1-12, 1993.

[6] J.J. Dongarra, R. Hempel, A.J.G. Hey, D.W. Walker. "A Proposal for a User-level, Message Passing Interface in a Distributed Memory Environment", Technical Report TM 12231, Oak Ridge National Laboratory, 1993.

[7] C. Dwork, M. Herlihy, O. Waarts, "Contention in Shared Memory Algorithms", *Proc. of 25-th ACM STOC*, pp. 174-183, 1993.

[8] P.B. Gibbons, "A More Practical PRAM Model," *Proc. of 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 158-168, 1989.

[9] S.E. Hambrusch, F. Hameed, and A. Khokhar, "A Study of Coarse-Grained Communication Operations on Mesh Architectures" Technical Report, Purdue University, May 1994.

[10] T. Heywood and S. Ranka, "A Practical Hierarchical Model of Parallel Computation: I. The model," *JPDC*, Vol. 16, pp. 212-232, 1992.

[11] P. Liu, W. Aiello, S. Bhatt, "An Atomic Model for Message Passing," *Proc. of ACM SPAA*, pp. 154-163, 1993.

[12] R. Ponnusamy, A. Choudhary, G. Fox, "Communication Overhead on CM5: An Experimental Performance Evaluation," *Proc. of 4-th Symp. on the Frontiers of Massively Parallel Computation*, pp. 108-115, 1992.

[13] R. Thakur, A. Choudhary, "All-to-all Communication on Meshes with Wormhole Routing," to appear in *Proc. of IPPS*, pp. 561-565, 1994.

[14] D.S. Scott, "Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies," *Proc. of 6-th Distributed Memory Computing Conference*, pp. 398-403, 1991.

[15] P. de la Torre and C.P. Kruskal, "Towards a Single Model of Efficient Computation in Real Parallel Machines," *Future Generation Computer Systems*, Vol. 8, pp. 395-408. 1992.

[16] L.G. Valiant, "A Bridging Model for Parallel Computation," *CACM*, 1990, Vol. 33, No. 8, pp. 103-111.

[17] D.S. Wills and W. Dally, "Pi: A Parallel Architecture Interface," *Proc. of 4-th Symp. on the Frontiers of Massively Parallel Computation*, pp. 345-352, 1992.