User Transparent Parallel Image Processing
Seinstra, F.J.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 5

A Communication Model for Automatic Decomposition of Regular Domain Problems*

"Mind the gap!"

(warning message broadcast across platforms at London Underground)

One of the most fundamental problems any automatic parallelization and optimization tool is confronted with is to find an optimal domain decomposition for an application at hand. For regular domain problems (such as simple matrix manipulations) this task may seem trivial. However, communication costs in programs executing on commodity clusters often significantly depend on the capabilities and particular behavior of the applied message passing primitives. As a consequence, straightforward domain decompositions may deliver non-optimal performance.

Whereas many software libraries exist that provide efficient message passing implementations [53, 102], MPI seems to have become the de facto standard [104]. Of the large number of functions defined in MPI 1.1, the two blocking point-to-point communication operations (i.e., MPI_Send() and MPI_Recv()) are most important and most often used (see also Section 2.2.1). To implement optimal parallel applications it is essential to have a thorough understanding of the performance characteristics of these basic communication operations. A good way to make such characteristics explicit is to design a performance model that captures typical point-to-point communication behavior. Because a fundamental MPI design criterion was portability across a wide

*This chapter combines our papers published in Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP 2002) [142], IEEE Transactions on Parallel and Distributed Systems [143] and Journal of Systems Architecture [144].
range of computers, such a model must be *applicable* to the same range of machines. Essentially, this implies that a performance model must incorporate a similar level of abstraction as introduced in the MPI standard.

In the literature several point-to-point communication models have been described that match the MPI abstractions up to a certain degree (e.g., the Postal Model [11, 21], LogP [38], and LogGP [1]). Although successful in many situations, these models were not designed for communication according to MPI specifically. Consequently, the models do not incorporate all capabilities of MPI's send and receive operations. As an important example, the effect of memory layout on communication costs is ignored completely. This is unfortunate, as the work of Prieto et al. [124, 125] indicates that a change in the spatial locality of messages exchanged using MPI can have a severe impact on the overall performance of an application. The authors state that "the bandwidth reduction due to non-unit-stride memory access could be more significant than the reduction due to contention in the network". Independently, we have come to similar conclusions [139]. Given these results, it is surprising that no model seems to exist that can account for such costs.

As described in Chapter 4, in our software architecture we rely heavily on performance models to perform the task of automatic parallelization of a particular class of regular domain problems: i.e., low level image processing. As the limitations of existing communication models proved to be too severe, we have designed a new model (called P-3PC, or the Parameterized model based on the Three Paths of Communication), that closely matches the behavior of MPI's standard point-to-point communication operations. P-3PC bears strong resemblance to the aforementioned models, but due to its additional features it provides more accurate estimations in many essential situations.

First, P-3PC acknowledges the difference in the time either the sender or the receiver is occupied in a message transfer, and the complete end-to-end delivery time. Second, P-3PC makes a distinction between communicating data stored either contiguously or noncontiguously in memory. Finally, P-3PC does not assume a strictly linear relationship between the size of a message being transmitted and the communication costs. Although P-3PC is targeted towards the specific needs in our research, it is general enough to be applicable in other research areas as well.

Hence, the primary research issue addressed in this chapter is formulated as follows: How to design a simple and portable communication model that (1) reflects the relevant capabilities of MPI's standard point-to-point communication primitives, and (2) accurately models the communication costs encountered in low level image processing applications executing in data parallel fashion.

This chapter is organized as follows. Section 5.1 discusses the requirements for a model to be applied in our software architecture. Also, two popular communication models are evaluated according to these requirements. The new P-3PC model is introduced in Section 5.2. Section 5.3 shows how P-3PC is applied in the evaluation of communication algorithms executed in a realistic image processing application. In Section 5.4 predictions are compared with results obtained on two clusters, each having a different interconnection network, and a different MPI implementation. Concluding remarks are given in Section 5.5.
5.1 Modeling of Message Passing Programs

In our software architecture all parallelization and optimization issues are to be taken care of automatically, hidden from the user. As explained in Chapters 2 and 4, for this task to be performed correctly we rely on domain-specific performance models that are applied in combination with a benchmarking tool and a separate scheduling component. Based on the models and the measured performance values, it is the task of the scheduler to make optimization decisions regarding:

1. the logical processor grid to map data structures onto (i.e., the actual domain decomposition),

2. the routing pattern for the distribution of data,

3. the number of processing units, and

4. the type of data distribution (e.g., broadcast instead of scatter).

In this chapter we focus on the first two optimization tasks in this list. Once the cost characteristics are available of any routing pattern, given any conceivable domain decomposition, the optimal number of processors and the actual type of data distribution can be derived.

In the following we will investigate the requirements for a communication model to be applied in our software architecture. On the basis of these requirements, we will shortly discuss the two most popular models described in the literature.

5.1.1 Model Requirements

In our software library all communication algorithms are implemented using the standard blocking MPI send and receive operations. Because low level image processing operations tend to have a bulk synchronous parallel behavior [103, 162], usage of any of MPI's additional communication modes will hardly result in a performance improvement, and may even be counterproductive (see also [125]). Also, as MPI’s standard collective communication operations do not provide all functionality required in our library\(^1\) we have implemented multiple scatter, gather, and broadcast operations in this manner as well.

In such data exchange operations the combined latency of sending or receiving multiple messages in sequence may be overlapping with the end-to-end latency of each single message. As shown in Figure 5.1(a), such latency differences can be significant. This overlap can be made explicit if a performance model incorporates the following properties:

1. The ability to predict the time a processing unit is busy executing either the `MPI_Send()` or the `MPI_Recv()` operation. As the two communicating nodes

\(^1\)The main problem with many of the operations defined in MPI 1.1 is that a possibility to define fluctuating strides in multiple dimensions is lacking. Although this problem is lifted in the MPI-2 definition [105] (with the introduction of the `MPI.Gatherv()` and `MPI.Scatterv()` operations), as of yet MPI-2 implementations are not generally available.
may handle the transfer of data differently (see [16], and also requirement 3 in this section), the communication costs at both ends should be modeled independently.

2. The ability to predict the complete end-to-end latency. Again, the end-to-end latency should be modeled independently from the overhead at either node.

Depending on the type of domain decomposition, it may be necessary to communicate data stored noncontiguously in memory. Using MPI derived datatypes it is possible to send such data in a single communication step. As was shown by Prieto et al. [124, 125], knowledge of a message's memory layout is important, as non-unit-stride memory access may have a severe impact on performance due to caching. In addition, the MPI send and receive operations may even handle the transmission of noncontiguous data differently from contiguous blocks. The MPI 1.1 definition [104] states that "it is up to the implementation to decide whether data should first be packed in a contiguous buffer before being transmitted, or whether it can be collected directly from where it resides". As shown in Figure 5.1(a) as well, the latency for communicating either contiguous or noncontiguous data may be significantly different indeed. Such differences can be accounted for if a performance model incorporates:

3. The ability to reflect the difference in sending data stored contiguously in memory, and noncontiguous data. Again, the memory layout at the two nodes should be modeled independently.

As a consequence from the fact that the send and receive operations are essentially 'black boxes', it is not safe to assume communication costs to be linearly dependent on message size. As shown in Figure 5.1(b), nonlinearities — caused by caching, buffering, packetization, changes in communication policy, etcetera — may be quite significant. As a final requirement, a model should therefore incorporate:

![Figure 5.1](image)

(a) Latency: sender side versus end-to-end

(b) Sender latency (detailed)

Figure 5.1: Values obtained on DAS [7] using MPI-LFC [16] (as in Section 5.4).
4. The ability to provide accurate predictions over a large range of message sizes. For the full range of message sizes a strictly linear increase in communication costs should not be assumed.

In certain application areas it may be important to incorporate network contention as well. For our purposes, however, this is not required. In Section 5.5 we will shortly come back to this issue.

5.1.2 Relevant Models in the Literature

In the literature a multitude of message passing models exists. One end of the spectrum consists of models in which communication costs are accounted for by abstracting the interconnection network into a few parameters (e.g., LogP [38], LogGP [1], the Postal Model [11, 21], and the standard linear communication model as described in [50, 79, 114]). Models with a similar level of abstraction are sometimes integrated in a model for computation in order to evaluate architecture and application scalability (e.g., the Latency Metric [173]). At the other end of the spectrum are highly parameterized models that are targeted towards a limited set of applications or architectures only (e.g., C^3 [63]).

In our research we must restrict ourselves to models that have an abstraction level comparable to that of MPI. Therefore, models such as the Postal Model, or LogP are seemingly most suitable. As is shown in the following, however, none of these models fully complies with the specific requirements in our research.

The Postal Model

One of the simplest point-to-point communication models is the Postal Model [11, 21], which derives its name from an analogy to the postal service. The model incorporates the notion of communication latency through a parameter $\lambda$, which represents the inverse ratio of the time it takes a processor to send out a message and the time until the recipient of the message has accepted it. As such, the single parameter captures both the software and the hardware related overhead, such as message preparation, local buffer copying, network propagation delays, and message interpretation. In the

![Diagram](image)

Figure 5.2: Communication according to the Postal Model.
model (see Figure 5.2) a *message* refers to an atomic piece of data, which cannot be broken into smaller pieces. The sending of large amounts of data is achieved by sending out several atomic messages in sequence. The time it takes to send or receive a message is defined as one unit of time.

The Postal Model partially adheres to the first two requirements of Section 5.1.1: it acknowledges the difference in the occupation time at each node, and the complete end-to-end latency. However, communication overhead is assumed to be identical at both ends. An assumption of this kind is overly restrictive and is a partial violation of the first two requirements of Section 5.1.1.

The model violates the third requirement as well as it does not allow changes in communication behavior induced by memory layout differences to be made explicit. In addition, the Postal Model uses a single unit time for the sending of atomic messages. This assumes a linear growth rate in the time required for sending messages of arbitrary length. This property does resemble the strategy of breaking down large messages into multiple packets (as applied by many message passing systems). However, it constitutes a violation of the fourth requirement of Section 5.1.1 as well.

**LogP and LogGP**

Another communication model that has received considerable attention is the LogP model [38]. The model captures the cost of communicating small-sized messages in four parameters:

- *L*: an upper bound to the *latency* associated with sending a message from one node to another.
- *o*: the *overhead*, or the amount of time a processor is busy during the transmission or reception of a message.
- *g*: the *gap*, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at the same processor.
- *P*: the number of processor-memory pairs in the machine.

Figure 5.3 shows that LogP bears strong resemblance to the Postal Model. Also, LogP presents a generalization of the Parameterized Communication Model [115], not discussed here.

![Figure 5.3: Communication according to LogP.](image-url)
In [1] the LogP model is extended with a linear model for long messages. This model, called LogGP, has one additional parameter:

- \( G \): the gap per byte (i.e., time per byte) for long messages.

A pictorial view of LogGP is given in Figure 5.4. Clearly, LogGP provides a more accurate description of the communication of long messages than a sequence of LogP communications.

The two models are important because they make explicit the differences in the occupation time at both ends, and the end-to-end delivery time. A possible delay (the gap \( g \)) in consecutive transmissions or receipts is accounted for as well. Unfortunately, the two models suffer from the same problems as the Postal Model. First, in both models communication overhead is assumed to be identical at both ends. Second, memory layout differences are not incorporated. Finally, the models assume a strictly linear growth rate in the time required for sending messages of arbitrary length. As a consequence, we conclude that the two models do not comply with the specific needs in our research either.

---

**5.2 The P-3PC Model**

As no model exists that meets all requirements of Section 5.1.1, we introduce a new communication model. The model, which we refer to as P-3PC, or the Parameterized model based on the Three Paths of Communication, will be discussed in two parts. First we introduce a simplified version of the complete model (called 3PC), that complies only with the first two requirements of Section 5.1.1. Subsequently, the 3PC model is extended to incorporate the remaining two requirements.

**5.2.1 Part I: 3PC**

Given the first two requirements of Section 5.1.1, we introduce the notion of the three paths of communication, and assume that the cost of message transmission can be captured in three independent values:
- $T_{\text{send}}$: the cost related to the communication path at the sender (i.e., the time required for executing the MPISend() operation).

- $T_{\text{recv}}$: the cost related to the communication path at the receiver (i.e., the time required for executing the MPIRecv() operation).

- $T_{\text{full}}$: the cost related to the full communication path (i.e., the time from the moment the sender initiates a transmission until the receiver has safely stored all data and is ready to continue).

For each path we assume that the communication costs can be represented by two parameters. The transmission of any message is expected to involve a constant amount of time, identical to the cost of sending a 0-sized message. This cost is captured by the mutually independent parameters $t_{cs}$, $t_{cr}$, and $t_{cf}$ (for the sender, receiver, and full path respectively). At the sender side this value may represent what is often referred to as the message startup time, but we prefer not to use this terminology to avoid unnecessary overspecification. Also, for each transmitted byte we assume an 'additional time', which is captured by the mutually independent parameters $t_{as}$, $t_{ar}$, and $t_{af}$ respectively. The three communication times (see also Figure 5.5) involved in the transmission of a message containing $n$ bytes are then given by:

$$
T_{\text{send}}(n) = t_{cs} + n \cdot t_{as},
$$
$$
T_{\text{recv}}(n) = t_{cr} + n \cdot t_{ar},
$$
$$
T_{\text{full}}(n) = t_{cf} + n \cdot t_{af}.
$$

Thus, 3PC simply constitutes a combination of three traditional linear models as also applied in [50, 79, 114]. Note that the manner in which accurate values for the model parameters can be obtained is independent of the actual MPI implementation or the type of communication hardware used. A detailed description of our method of measurement is given in Section 5.4.

![Figure 5.5: Communication according to 3PC.](image-url)
5.2.2 3PC versus LogGP

The LogGP model of Figure 5.4 constitutes a superset of all conventional models of Section 5.1.2. In other words, it is possible to express models such as the Postal Model, or LogP, in terms of the LogGP parameters. For this reason it is relevant to indicate that 3PC preserves the important qualities of LogGP under the following assumptions:

\[ t_{cs} = t_{cr} = g, \]
\[ t_{af} = 2o + L, \]
\[ t_{as} = t_{ar} = t_{af} = G. \]

Because in state-of-the-art communication processors LogGP's \( o \) parameter is either negligible [22] or comparable to \( g \) (even for relatively small messages, see [92]), 3PC is even identical to LogGP under the given assumptions. Compared to LogGP we feel that 3PC is easier to understand, as for each communication path similar parameters are defined. Given the fact that the costs for the three paths of communication are made independent (which is not the case in any of the other models), we conclude that 3PC is expected to be at least as powerful as the LogGP model. Note, however, that we do not claim that 3PC is necessarily a better alternative to LogGP for detailed study of communication behavior. It is introduced only for it to serve as a basis for the P-3PC model.

5.2.3 Part II: P-3PC

To incorporate the last two requirements of Section 5.1.1, the 3PC model is 'parameterized' with a cost indicator \( M \), representing the memory layout at the two communicating nodes. Also, it is assumed that each 'additional time' parameter is a function of \( n \), instead of a constant value for all message sizes. In this extended model (called \( P-3PC \)), the three communication times involved in a message transfer are given by:

\[ T_{\text{send},M}(n) = t_{cs} + t_{as,M}(n), \]
\[ T_{\text{recv},M}(n) = t_{cr} + t_{ar,M}(n), \]
\[ T_{\text{full},M}(n) = t_{cf} + t_{af,M}(n), \]

where \( M \in \{cc, cn, nc, nn\} \). These layout descriptors indicate the four memory layout combinations at the sender and the receiver combined. For example, \( cn \) means that a contiguous block of data is transmitted by the sender, which is accepted as a noncontiguous block by the receiver.

As no a priori assumptions can be made about the shape of the 'additional time' functions, a set of benchmarking operations must be performed for several different message sizes. As also indicated in Chapter 4, one possibility is to arbitrarily choose a set of relevant message sizes, but an adaptive benchmarking technique could be used as well to actively search for nonlinearities in the communication costs. In any case, based on the benchmarking results (and in accordance with the fourth requirement of Section 5.1.1), each 'additional time' function is assumed to be piecewise linear between each pair of measured communication cost values.
5.3 Application of the P-3PC Model

This section shows how the P-3PC model is applied to evaluate the communication costs involved in one of the most essential applications in image processing: i.e., evaluation of the differential structure of images. Examples are edge detection (based on first and second order derivatives) and invariants (based on i-th order derivatives). Applications of this kind are good examples of regular domain problems as referred to in the work of Prieto et al. [124, 125].

As is well-known, a derivative is best computed using convolution with a separable Gaussian kernel (i.e., n 1-D kernels, each applied in one of the image’s n dimensions). The size of the convolution kernel depends on the smoothing scale \( \sigma \) and the order of the derivative. In this example (and in the measurements discussed in the next section) we restrict ourselves to first and second order derivatives (five in total) in the \( x- \) and \( y- \) direction of 2-D image data, and \( \sigma \in \{1, 3, 5\} \). Here, for \( \sigma = 1 \), the sizes of the 1-D kernels for the \( i \)-th order derivative (with \( i \in \{0, 1, 2\} \)) in any direction are 7, 9, and 9 pixels respectively. For \( \sigma = 3 \) the kernel sizes are 15, 23, and 25 pixels, and for \( \sigma = 5 \) these are 23, 37, and 39 pixels respectively. For readers unfamiliar with image processing it is sufficient to know that these kernel sizes partially determine the amount of data exchanged among neighbors in a logical CPU grid — as is explained in more detail below.

When running such application in parallel, three different communication algorithms are to be executed. First, the input image is to be spread throughout the parallel system in a scatter operation. Second, to calculate partial derivative images, pixels in the border regions of each partial input image are to be exchanged among neighboring nodes in the logical CPU grid. Finally, after having performed all relevant (application dependent) sequential operations, resulting image data is to be gathered at a single node, for on-screen display or storage.

![Figure 5.6: Comparison of MPLScatterv() and OFT scatter implemented using MPLSend() and MPLRecv() calls (measured on DAS using MPI-LFC).](image)

Figure 5.6: Comparison of MPLScatterv() and OFT scatter implemented using MPLSend() and MPLRecv() calls (measured on DAS using MPI-LFC).
As indicated in Section 5.1.1, in addition to the collective operations available in MPI we have implemented multiple scatter and gather operations ourselves, using standard blocking point-to-point operations. As shown in Figure 5.6, our implementations — which, in contrast to the MPI versions, allow definition of fluctuating strides in multiple dimensions — can often compete with available MPI implementations. This indicates that many MPI distributions are not optimized for a particular machine, a problem also discussed in [125, 163]. Of course, in cases where the MPI implementations are faster (and match our specific needs), we apply these versions and use the P-3PC estimations for our fastest implementation as an upper bound. In the following, the modeling of such operations is restricted to two different implementations, one based on a one-level flat tree (OFT), and the other based on a spanning binomial tree (SBT) (see Figure 5.7).

In case of the OFT scatter operation the root sends out data to all other nodes in sequence. If a $1 \times P$ logical CPU grid is assumed (where $P$ is the number of nodes), for each node the data sent out by the root is stored contiguously in memory; for all other grids all data blocks sent out are noncontiguous. In addition, for all possible grids all data is accepted as a contiguous block at each receiving node. As each node in the OFT has to wait for all lower-numbered nodes to be serviced by the root before it will receive data itself, the communication costs are highest at either the root or at the leaf node that is last serviced (depending on the benchmarking results). A worst case P-3PC estimation of this operation is shown in the timeOFTscatter() operation in Listing 5.1. An estimation of the related OFT gather operation is simply obtained by setting $nc$ to $cn$, and changing all occurrences of $T_{send}$ to $T_{recv}$.

P-3PC estimation of the spanning binomial tree scatter operation is slightly more complicated. In such operation the root node sends out data to $\log P$ other nodes. Also, each non-leaf node forwards all received data it is not responsible for. If $X$ is the number of nodes defined in the $x$-direction of the logical CPU grid, the number of messages involving contiguous data blocks sent out by the root is $\log P - \log X$; the remaining messages sent out are all noncontiguous. In general, the communication costs will be highest at either the root node, or the node that is $\log P$ full communication paths away from the root. The timeSBTscatter() operation in Listing 5.1 shows the worst case P-3PC estimation of this operation. An estimation of the related SBT gather operation is obtained as before.

![Example communication trees for data scattering.](image)
A well-known method to implement Gaussian convolution is to extend the domain of the image structure with a scratch border that, on each side of the image in dimension \( n \), has a size of about half the 1-D kernel applied in that dimension. When executed in parallel, neighboring nodes in the logical CPU grid need to exchange pixel values to correctly fill the borders of all extended partial images. In our library, the exchange of border data is executed in four communication steps. First, each node sends a subset of its local partial image to the neighboring node on its right side in the logical CPU grid (if such neighbor exists). When a node has accepted this block of data (i.e., after a full communication path period), it subsequently transmits a subset of its local partial image to its left neighbor. As shown in Figure 5.8 these steps in the border exchange algorithm always involve noncontiguous blocks of data. Similarly, in the next two steps border data is exchanged in upward and downward direction, in both cases involving contiguous blocks only. Thus, the \texttt{timeBorderExchange()} operation in Listing 5.1 gives a worst case P-3PC estimation for this routine.

Listing 5.1: \textit{P-3PC estimation of OFT \& SBT scatter, and border exchange.}

```c
double timeOFTscatter() {
    M = (X == 1) ? \( cc : nc \) // \( X = \text{nr. of nodes in x-direction of logical CPU grid} \)
    time1 = (\( P - 1 \)) \cdot T_{\text{send,cc}}(\text{imw} \cdot \text{imh}/P) // \( P = \text{total nr. of nodes} \)
    time2 = (\( P - 2 \)) \cdot T_{\text{send,cc}}(\text{imw} \cdot \text{imh}/P) + T_{\text{full,cc}}(\text{imw} \cdot \text{imh}/P) // \text{imw = image width} \text{imh = image height} 
    return max(time1, time2)
}

double timeSBTscatter() {
    time1 = 0.0
    time2 = 0.0
    for (i=1; i<=\log P - \log X; i++)
        time1 = time1 + T_{\text{send,cc}}(\text{imw} \cdot \text{imh}/(2 \cdot i))
        time2 = time2 + T_{\text{full,cc}}(\text{imw} \cdot \text{imh}/(2 \cdot i))
    }
    for (i=\log P - \log X + 1; i<=\log P; i++)
        time1 = time1 + T_{\text{send,cc}}(\text{imw} \cdot \text{imh}/(2 \cdot i))
        time2 = time2 + T_{\text{full,cc}}(\text{imw} \cdot \text{imh}/(2 \cdot i))
    }
    return max(time1, time2)
}

double timeBorderExchange() {
    return (2 \cdot T_{\text{full,nn}}(\text{bw} \cdot \text{imh}) + 2 \cdot T_{\text{full,cc}}((\text{imw} + 2 \cdot \text{bw}) \cdot \text{bh})) // \text{bw = border width} \text{bh = border height} 
}
```

Figure 5.8: Border exchange (right-left and down-up).
5.4 Measurements and Validation

To validate the P-3PC model we have performed a representative set of benchmarking operations. For each communication path and memory layout combination measurements were performed using 4 different message sizes, arbitrarily set at 1K, 50K, 100K and 500K (all 4-byte values). Benchmarking was performed for 0-sized messages as well. Note that these values are not chosen to best match the communication characteristics for one particular parallel computer. These sizes are representative for messages transmitted in many image processing applications, and are set identically for all machines. Also note that the sizes applied by the architecture’s benchmarking tool can be user-defined as well; the sizes given here are used by default.

Clearly, there is a trade-off between the number of benchmarking operations to be performed and the obtainable estimation accuracy. Still, the predefined set of only 4 message sizes is generally sufficient to obtain highly accurate performance estimations for the much larger range of message sizes encountered in a real application. In this respect it is important to note that, in the measurements presented in the remainder of this chapter, actual message sizes range from 192 bytes up to 8 MB.

```c
double timePath(int pathType, int bufsize, int sendLayout, int recvLayout, int nrRounds) {
    if (sendLayout .eq. NONCONTIGUOUS) // definition of 'sendType'
        MPI.Type-vector(100, bufsize/100, 2*bufsize/100, MPI.FLOAT, &sendType);
    else
        MPI.Type-vector(1, bufsize, bufsize, MPI.FLOAT, &sendType);
    // definition of 'recvType' is similar

    for (i=1:nrRounds) {
        if (myCPU().eq. 0) {
            if (pathType .eq. SEND) { // measure send path
                time1 — MPI.Wtime();
                MPI.Send(buf, 1, sendType, 1, ...);
                time2 — MPI.Wtime();
                total — total + time2-time1;
            } else if (pathType .eq. RECV) { // measure receive path
                time1 — MPI.Wtime();
                MPI.Recv(buf, 1, recvType, 1, ...);
                time2 — MPI.Wtime();
                total — total + time2-time1;
            } else if (pathType .eq. FULL) { // measure full path
                time1 — MPI.Wtime();
                MPI.Send(buf, 1, sendType, 1, ...);
                MPI.Recv(buf, 0, recvType, 1, ...);
                time2 — MPI.Wtime();
                total — total + ((bufsize .eq. 0) ? (time2-time1)/2 : (time2-time1)-2-tcf);
            }
        } else if (myCPU().eq. 1) // matching send and recv calls at node 1 are not shown
            return (total/nrRounds);
    }
}
```

Listing 5.2: Pseudo code for benchmarking all path-layout combinations. The constant time values \(t_{cs}\), \(t_{cr}\), and \(t_{cf}\) are obtained if bufsize equals equals zero.
To give full insight in the benchmarking process, Listing 5.2 gives a simplified overview in pseudo code. To measure communication for noncontiguous data, a fixed number of 100 memory blocks (a conservative estimate of the number of blocks possibly used in a real application, and again a default setting) is combined in a single derived datatype definition. For contiguous data only one block is used in such definition. Measurements for the send and receive paths are obtained by letting one node continuously send data to another node. Full communication path measurements are obtained by subsequently sending out a message of size 'bufsize', and receiving a 0-sized message. As these operations are similar to those applied by many others in the literature we leave all further interpretation to the reader.

5.4.1 Distributed ASCI Supercomputer (DAS)

The first set of measurements was performed on the 128-node homogeneous DAS-cluster [7] located at the Vrije Universiteit in Amsterdam. All measurements were performed using MPI-LFC [16], an implementation which is partially optimized for the DAS. The 200 Mhz Pentium Pro nodes (with 128 MByte of EDO-RAM) are connected by a 1.2 Gbit/sec full-duplex Myrinet network, and run RedHat Linux 6.2.

The performance values obtained for this machine are presented in Figure 5.9. The values indicate that transmitting noncontiguous data indeed has a significant impact on performance. In this case, the additional overhead is due to the fact that MPI-LFC uses a contiguous send-buffer for noncontiguous data. To preserve the elegance of the benchmarking code, we have measured multiple 'constant time' values for each communication path \( (m = 0) \). These additional values do not affect the estimations presented in this section in any way.

In the following we show the results as obtained for the example application of Section 5.3. For each of the communication algorithms we have been careful to keep

<table>
<thead>
<tr>
<th></th>
<th>m=0</th>
<th>m=1K</th>
<th>m=50K</th>
<th>m=100K</th>
<th>m=500K</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{send,cc} )</td>
<td>5.98</td>
<td>61.72</td>
<td>4355.45</td>
<td>10246.77</td>
<td>58596.98</td>
</tr>
<tr>
<td>( T_{send,cn} )</td>
<td>8.04</td>
<td>60.74</td>
<td>4363.35</td>
<td>9853.95</td>
<td>57141.29</td>
</tr>
<tr>
<td>( T_{send,nc} )</td>
<td>7.93</td>
<td>248.88</td>
<td>5722.00</td>
<td>15142.74</td>
<td>90478.81</td>
</tr>
<tr>
<td>( T_{send,nn} )</td>
<td>8.29</td>
<td>133.88</td>
<td>5582.23</td>
<td>14137.45</td>
<td>87870.27</td>
</tr>
<tr>
<td>( T_{recv,cc} )</td>
<td>14.86</td>
<td>58.08</td>
<td>5754.93</td>
<td>12037.78</td>
<td>60062.70</td>
</tr>
<tr>
<td>( T_{recv,cn} )</td>
<td>14.89</td>
<td>127.30</td>
<td>9527.59</td>
<td>19467.08</td>
<td>98016.47</td>
</tr>
<tr>
<td>( T_{recv,nc} )</td>
<td>14.43</td>
<td>46.56</td>
<td>5517.28</td>
<td>12364.45</td>
<td>61446.05</td>
</tr>
<tr>
<td>( T_{recv,nn} )</td>
<td>14.82</td>
<td>125.05</td>
<td>9340.63</td>
<td>19685.86</td>
<td>98275.11</td>
</tr>
<tr>
<td>( T_{full,cc} )</td>
<td>23.61</td>
<td>131.39</td>
<td>4506.32</td>
<td>11007.89</td>
<td>61277.46</td>
</tr>
<tr>
<td>( T_{full,cn} )</td>
<td>25.54</td>
<td>214.10</td>
<td>8665.39</td>
<td>19195.53</td>
<td>97219.23</td>
</tr>
<tr>
<td>( T_{full,nc} )</td>
<td>27.05</td>
<td>206.94</td>
<td>6696.30</td>
<td>18015.91</td>
<td>95546.60</td>
</tr>
<tr>
<td>( T_{full,nn} )</td>
<td>24.47</td>
<td>287.89</td>
<td>11746.29</td>
<td>25652.54</td>
<td>132399.20</td>
</tr>
</tbody>
</table>

Figure 5.9: Benchmarking results obtained on DAS (in \( \mu s \)).
the intrusiveness of the measurements to a minimum. All P-3PC estimations are obtained as in Listing 5.1. Also, in all situations we compare our results with those obtained with LogGP. To avoid using a particularly bad value for the 'G' parameter, we assume a piece-wise linear dependence on message size in the LogGP model as well. In addition, to be able to use the measured values of Figure 5.9, we have reduced the P-3PC model into LogGP in the following manner: \( g = t_{ca}, \quad L = t_{cf}, \quad G = t_{af,cc}. \)

As indicated in Section 5.2.2, this reduction makes P-3PC identical to LogGP. Still, to overcome any problem the reader may have with this interpretation of the model, in the remainder we will refer to it as LogGP*.

In Figure 5.10(a) results are presented for a \( 512^2 \) floating point image, which is mapped onto a \( 1 \times 16 \) logical CPU grid. The graph shows results for the two available implementations of the scatter and gather routines, as well as for the border exchange (for all \( \sigma \in \{1,3,5\} \)). For such data decomposition all messages involve contiguous blocks only. This is even the case for the border exchange, as no node has a neighbor to its left or right. The graph shows that P-3PC and LogGP* are both quite accurate for this type of data decomposition. As was to be expected, the estimations obtained from the two models are comparable, although P-3PC seems to do marginally better. Apparently, introduction of the three communication paths indeed produces a slightly more accurate model. Here, the differences are marginal, however, and provide no justification for P-3PC's added complexity.

As can be seen in Figure 5.10(b), for a \( 16 \times 1 \) data decomposition P-3PC outperforms LogGP* by far. This is because for such decomposition all messages involve noncontiguous data at the sender side. Figure 5.10(c) and Figure 5.10(d) show similar results for \( 8 \times 1 \) and \( 32 \times 1 \) decompositions. A comparison for larger image data structures is shown in Figure 5.10(e) and Figure 5.10(f). Although most P-3PC estimations are highly accurate, deviations from actual measurements are usually due to small inaccuracies in the performance values obtained by benchmarking. Sometimes, algorithm performance is also slightly degraded by contention in the network — an effect not accounted for by P-3PC. However, the impact of memory layout on performance is always more significant than that of contention. Note that this matches the results of [124, 125].

Figure 5.10(g) and Figure 5.10(h) show that the P-3PC model indeed allows the scheduler of Section 5.1 to make correct optimization decisions. According to the LogGP* model, scattering or gathering a \( 256^2 \) floating point image is about as expensive for each communication tree and data decomposition. In practice this is not true, however, and P-3PC gives much more accurate estimations at all times.

Figure 5.11 gives results for the communication algorithms applied to all possible decompositions involving 16 nodes. Again, P-3PC outperforms LogGP* in almost all situations. It is interesting to see in Figure 5.11(a-d) that, while for all but the \( 1 \times 16 \) decomposition P-3PC is somewhat pessimistic, the estimations get better for decompositions that are 'closer' to \( 16 \times 1 \). This is explained by the fact that in the benchmarking phase noncontiguous communication is measured using blocks that have quite a significant distance from one another in memory. Thus, caching can become a significant factor, which is indeed expected to be most prominent in a \( 16 \times 1 \) decomposition (again, see also [124, 125]).
Figure 5.10: Measurements (DAS) versus P-3PC & LogGP* estimations (1).
Figure 5.11: Measurements (DAS) versus P-3PC & LogGP* estimations (2).

Figure 5.11(e) and Figure 5.11(f) show that P-3PC gives accurate estimates for the border exchange algorithm for all data decompositions as well. Whereas LogGP* indicates that a 4×4 decomposition is always optimal (which is explained by the fact that the amount of border data is smallest when each partial image is square), P-3PC correctly prefers the 2×8 decomposition. Because the exchange of border data may be performed hundreds of times in a realistic application (for example, see [55] for such application that even applies values of σ > 5), these results are important indeed. For additional results obtained on the DAS (also including sequential computation) we refer to [147].
5.4.2 Beowulf at SARA

The second set of tests was performed on the 40-node Beowulf-cluster located at SARA, Amsterdam. On this machine, measurements and benchmarking were performed using MPICH-1.2.0 [61]. The 700 Mhz AMD Athlon nodes (with 256 MByte of RAM) are connected by a 100 Mbit/sec switched Ethernet network, and run Debian Linux 2.2.17.

Because the cluster is heavily used for other research projects as well, we have been able to use only 8 nodes at a time. Figure 5.12 presents results for all algorithms, using a $512^2$ floating point image which is mapped onto a $1\times8$ grid as well as a $8\times1$ grid. The graphs show that the two models are both quite good in all cases, but P-3PC again provides more accurate estimations. It is clear that the MPICH implementation is much better than the MPI-LFC implementation used on the DAS. Any additional overhead due to non-unit-stride memory access is not caused by buffer copying, but can be attributed to caching alone. Although less significant on the cluster at SARA, this is exactly the effect Prieto et al. have shown to be important on other parallel machines [124, 125].

![Graphs showing measurements versus P-3PC and LogGP* estimations for Beowulf at SARA.](image)

Figure 5.12: Measurements (Beowulf at SARA) versus P-3PC and LogGP* estimations.

5.5 Conclusions

In this chapter we have presented the new P-3PC model for predicting the execution time of communication algorithms implemented using MPI's standard point-to-point operations. P-3PC incorporates the notion of the 'three paths of communication', and accounts for differences in performance at the sender, the receiver, and the full communication path. In addition, P-3PC models the impact of memory layout on communication costs, and accounts for costs that are not linearly dependent on message size. Compared to similar models, P-3PC has the potential for higher predictive accuracy due to its close match with the capabilities and possible behavior of MPI's point-to-point operations.
P-3PC's predictive power is essential to perform the important task of automatic and optimal decomposition of regular domain problems. Although designed for this specific task, we expect the model to be relevant in other research areas as well. It is important to note, however, that P-3PC suffers from the same problem as other models that abstract from the actual network topology (see also [38]). The model can not discriminate between algorithms that cause severe network contention, and those that do not. In our research this is not a problem, as we only apply communication patterns that are expected to perform well on most network topologies used today. Still, because P-3PC is similar to the LogGP model, it can easily be extended to account for contention, in the same manner as described in [3].

It should also be noted that we do not claim the P-3PC model to give a precise characterization of all types of memory access. Any cost factors other than those related to contiguous and noncontiguous memory access are implicit (such as specific cache behavior, differences between programmed I/O and DMA transfer, etcetera), but are still captured due to the semi-empirical modeling approach described in Section 4.2. In this respect, an extension to the P-3PC model that would give a more detailed characterization of non-unit-stride memory access, would be to incorporate a stride parameter that captures the actual distances between contiguous blocks transmitted in a single communication step. We have not included such parameter as the results obtained with the current model were shown to be sufficiently accurate.

As the P-3PC model stresses the importance of benchmarking to obtain accurate values for the model parameters, one may argue that the predictive power of the model is limited. However, the model does not specifically enforce a large number of measurements to be performed. As for models that incorporate a similar level of abstraction, a set of three or four measurements for each communication path may already be sufficient to obtain accurate predictions. The P-3PC model merely acknowledges that nonlinearities in communication costs may be significant (as shown in Section 5.1.1) and should be accounted for.

We are aware of the fact that an evaluation of P-3PC is never complete. However, the evaluation as presented in this chapter — incorporating two fundamentally different interconnection networks, and two different MPI implementations — has shown the model to be highly accurate in estimating the communication costs related to any type of domain decomposition used in a realistic image processing application. As such, we have shown P-3PC to be useful as a basis for automatic and optimal decomposition within the extensive application area of regular domain problems. Also, because P-3PC is capable of modeling behavior that was shown to be problematic in [124, 125], we expect the model to be applicable to the very same machines and MPI implementations as well.