P-3PC: A Point-to-Point Communication Model for Automatic and Optimal Decomposition of Regular Domain Problems

F.J. Seinstra and D. Koelma

Abstract—One of the most fundamental problems automatic parallelization tools are confronted with is to find an optimal domain decomposition for a given application. For regular domain problems (such as simple matrix manipulations), this task may seem trivial. However, communication costs in message passing programs often significantly depend on the memory layout of data blocks to be transmitted. As a consequence, straightforward domain decompositions may be nonoptimal. In this paper, we introduce a new point-to-point communication model (called P-3PC, or the "Parameterized model based on the Three Paths of Communication") that is specifically designed to overcome this problem. In comparison with related models (e.g., LogGP) P-3PC is similar in complexity, but more accurate in many situations. Although the model is aimed at MPI's standard point-to-point operations, it is applicable to similar message passing definitions as well. The effectiveness of the model is tested in a framework for automatic parallelization of low level image processing applications. Experiments are performed on two Beowulf-type systems, each having a different interconnection network, and a different MPI implementation. Results show that, where other models frequently fail, P-3PC correctly predicts the communication costs related to any type of domain decomposition.

Index Terms—MPI, point-to-point communication, performance optimization, performance modeling, automatic domain decomposition.

1 Introduction

Message passing is used widely in software designed for execution on distributed memory MIMD-style multicomputers. Whereas, many software libraries exist that provide efficient message passing implementations [9], [15], MPI seems to have become the de facto standard [17]. 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.

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 range of computers, such model must be *applicable* to the same range of machines as well. Essentially, this means 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 [4], [6], LogP [7], and LogGP [1]). Although successful in many situations, these models do not incorporate all capabilities of MPI's send and receive operations. Most importantly, the effect of memory layout on communication costs is ignored completely. This is unfortunate, as the recent work of Prieto et al. [20], [21] 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. In this work, it is stated that "*the bandwidth reduction due to nonunit-stride memory access could be more significant than the reduction due to contention in the network*." Independently, we have come to similar conclusions [22]. Given these results, it is surprising that no model seems to exist that can account for such costs.

In our research, we rely heavily on performance models to perform the task of automatic parallelization of a particular class of regular domain problems, i.e., that of low level image processing [24]. As the limitations of existing communication models proved to be too severe, we have designed a new model (called *P-3PC*) that closely matches the behavior of MPI's standard point-to-point operations. P-3PC bears a strong resemblance to the aforementioned models, but due to its additional features it is capable of providing 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

• *The authors are with the Intelligent Sensory Information Systems, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands. E-mail: {fjseins, koelma}@science.uva.nl.*

Manuscript received 1 Apr. 2001; revised 21 Aug. 2001; accepted 5 Mar. 2002.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 114020.

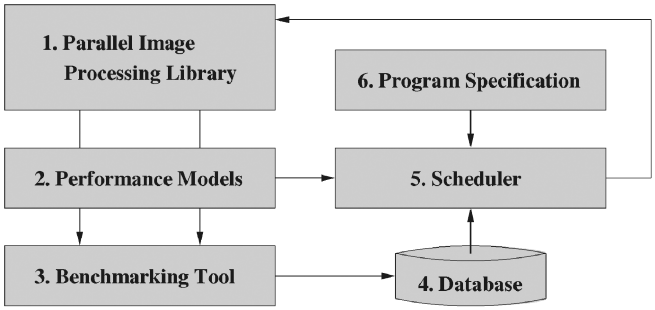

Fig. 1. Simplified architecture overview.

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 toward the specific needs in our research, it is general enough to be applicable in other research areas as well.

This paper is organized as follows: Section 2 discusses the requirements for a model to be applied in our research. The new P-3PC model is introduced in Section 3. Section 4 shows how P-3PC is applied in the evaluation of communication algorithms executed in a realistic image processing application. In Section 5, predictions are compared with results obtained on two Beowulf-type systems, each having a different interconnection network, and a different MPI implementation. Concluding remarks are given in Section 6.

2 Model-Based Domain Decomposition

The main objective in our research is to build a software architecture that allows image processing researchers to implement parallel applications in a transparent (i.e., sequential) manner [24]. All parallelization and optimization issues are to be taken care of by the architecture itself, hidden from the user. The architecture consists of six logical components (see Fig. 1). The first component is an extensive library of data parallel low level image processing operations. Each library operation is annotated with a performance model for runtime cost estimation. Essentially, one class of models deals with the costs of sequential computation only, and another class of models—which is the main topic of this paper—deals with the costs of interprocess communication. The third architectural component performs a set of benchmarking operations to obtain appropriate values for the model parameters for a given parallel machine. The benchmarking results are stored in a database of performance values, which, in turn, are used by a scheduling component to obtain an optimal schedule for a given image processing application. Essentially, the scheduler uses the models and the measured performance values to make optimizations 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).

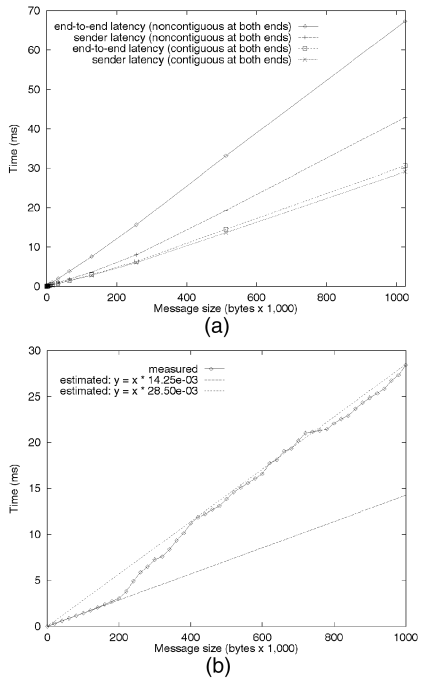

Fig. 2. Values obtained on DAS [3] using MPI-LFC [5] (as in Section 5).

The scheduling results are routed back to the image processing library to allow each operation to adapt (optimize) its behavior at runtime.

2.1 Model Requirements

In our 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 [16], [26], usage of any of MPI's additional communication modes will hardly result in a performance improvement, and may even be counterproductive (see also [21]). 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 Fig. 2a, 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 may handle the transfer of data differently (see [5], and also requirement 3 in this section), the communication costs at both ends should be modeled independently.

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 [18] (with the introduction of the `MPI_Gatherw()` and `MPI_Scatterw()` operations), as of yet MPI-2 implementations are not generally available.

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. [20], [21], knowledge of a message's memory layout is important, as nonunit-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 [17] 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 Fig. 2a 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 Fig. 2b, nonlinearities—caused by caching, buffering, packetization, changes in communication policy, ect.—may be quite significant. As a final requirement, a model should therefore incorporate:

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 6, we will shortly come back to this issue.

2.2 Relevant Models in the Literature

In the literature, a multitude of message passing models exist. 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 [7], LogGP [1], the Postal Model [4], [6], and the standard linear communication model as described in [8], [13], [19]). 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 [28]). At the other end of the spectrum are highly parameterized models that are targeted toward a limited set of applications or architectures only (e.g., C^3 [12]).

As indicated in Section 1, in our research we must restrict ourselves to models that have an abstraction level that is at least as high as that of MPI. Therefore, models such as LogP, the Postal Model, or the Latency Metric would be most suitable. However, none of these models fully complies with the specific needs in our research. This is because in all such models:

1. communication overhead is assumed to be identical at both ends, and/or
2. the impact of memory layout on communication costs is not incorporated, and/or
3. performance growth is assumed to be strictly linear (for a complete overview, see [23]).

3 The P-3PC Model

As no model exists that meets all requirements of Section 2.1, 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 2.1. Subsequently, the 3PC model is extended to incorporate the remaining two requirements.

3.1 Part I: 3PC

Given the first two requirements of Section 2.1, 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_{send}: the cost related to the communication path at the sender (i.e., the time required for executing the `MPI_Send()` operation).
- T_{recv}: the cost related to the communication path at the receiver (i.e., the time required for executing the `MPI_Recv()` operation).
- T_{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 (also, see Fig. 3a) involved in the transmission of a message containing n bytes are then given by:

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

Thus, 3PC simply constitutes a combination of three traditional linear models as also applied in [8], [13], [19]. Note that the manner in which accurate values for the model parameters can be obtained is independent of the

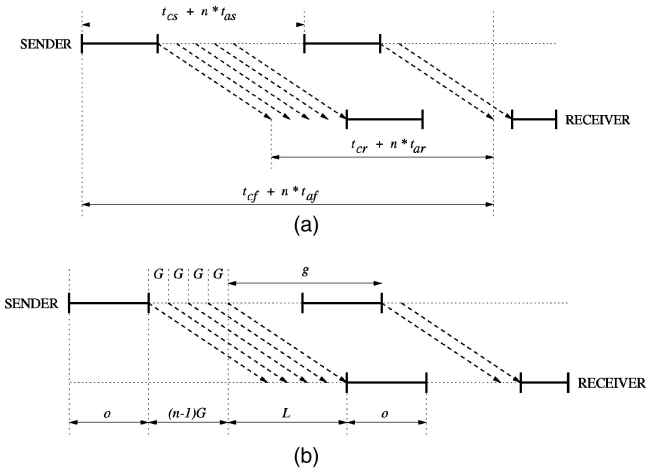

Fig. 3. Communication according to (a) 3PC and (b) LogGP.

actual MPI implementation or the type of communication hardware used. A detailed description of our method of measurement is given in Section 5.

3.2 3PC versus LogGP

The popular LogGP model (see Fig. 3b) constitutes a superset of all conventional models of Section 2.2. 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:

$$t_{cs} = t_{cr} = g,$$
$$t_{cf} = 2o + L,$$
$$t_{as} = t_{ar} = t_{af} = G.$$

Because LogGP's o parameter tends to be equal to g (even for relatively small messages, see [14]), 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.

3.3 Part II: P-3PC

To incorporate the last two requirements of Section 2.1, 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:

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

where $\mathtt{M} \in \{cc, cn, nc, nn\}$. 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. 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 2.1), each "additional time" function is assumed to be piecewise linear between each pair of measured communication cost values.

3.3.1 Semiempirical Modeling

It is important to note that the *semiempirical modeling* approach described above is similar to that of Xu et al. [27], in that essential but implicit cost factors are incorporated by performing actual experiments on a small set of sample data. To be more specific, in [27] Xu et al., follow a two-level hierarchical modeling approach to predict the performance of full applications. At the higher level, a *thread graph* is constructed that characterizes an application's runtime behavior. More importantly, at the lower-level, the elapsed times of individual code segments and events (e.g., explicit communication steps) are estimated using a combination of analytical and experimental methods. Essentially, each code segment is treated as a loop construct with a certain number of iterations. By measuring such loop construct with a small number of iterations (and given a set of application parameters), the performance of the same loop with an arbitrary number of iterations can be estimated. Explicit communication events are measured directly—in such a way that a distinction can be made between different types of communication events as present in operations such as barrier synchronization and locking. Based on these measurements, the performance of full applications can be estimated by adding the (interpolated) costs related to each code segment and communication event.

This strategy is quite similar to the approach followed in our research. Our higher level model consists of a so-called *application state transition graph* (dissimilar to the aforementioned thread graph), which is traversed to find parallel code with (expected) minimal execution time. At the lower-level we use benchmarking to obtain the performance characteristics of a set of high-level instructions that is defined in combination with an abstract machine for parallel image processing (the *APIPM*, see [24]). Most instructions that constitute image processing tasks resemble the loop constructs as used by Xu et al. (albeit for a more restricted application domain). The instructions related to communication resemble the aforementioned communication events. Due to the fact that our models are specifically

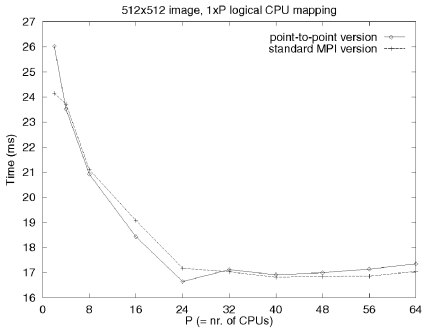

Fig. 4. Comparison of MPI_Scatterv() and OFT scatter implemented using MPI_Send() and MPI_Recv() calls (measured on DAS using MPI-LFC).

intended for MIMD-style multicomputers (as opposed to the models defined in [27]), and also because we focus our modeling on regular communication patterns only, we treat each possible communication event (i.e., a single point-to-point communication step) as identical. Consequently and in contrast to Xu et al., one of the cost factors that we do not incorporate (implicitly) is contention—an overhead which still may be significant for collective communication operations that are never used in our work. For a more detailed description of the benchmarking techniques that we apply for point-to-point communication operations, we refer to Section 5.

4 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. [20], [21].

As is well-known, a derivative is best computed using convolution with a separable Gaussian kernel (i.e., n 1D kernels, each applied in one of the image's n dimensions). The size of the convolution kernel depends on the smoothing scale σ 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 2D image data, and $\sigma \in \{1, 3, 5\}$. Here, for $\sigma = 1$, the sizes of the 1D kernels for the ith 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 an 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.

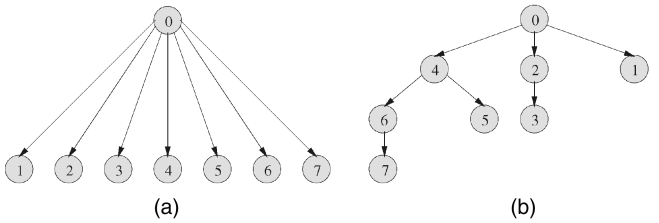

Fig. 5. Example communication trees for data scattering. (a) One-level flat tree and (b) spanning binomial tree.

As indicated in Section 2.1, 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 Fig. 4, 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 [21], [25]. 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 Fig. 5).

In case of the OFT scatter operation, the root node 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 in the tree), 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 leaf 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 node or at the leaf node that is last serviced (depending on the related benchmarking results). A worst-case P-3PC estimation of this operation is shown in the `timeOFTscatter()` operation in Fig. 6. 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 an operation, the root node sends out data to $logP$ other nodes. Also, each nonleaf 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 $logP - logX$; 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 $logP$ full communication paths away from the root. The `timeSBTscatter()` operation in Fig. 6 shows the worst-case P-3PC estimation of this

```
double timeOFTscatter() {
    M ← (X .eq. 1) ? cc : nc                    // X = nr. of nodes in x-direction of logical CPU grid
    time1 ← (P − 1) · T_send,M(imw · imh/P)      // P = total nr. of nodes
    time2 ← (P − 2) · T_send,M(imw · imh/P) + T_full,M(imw · imh/P)   // imw = image width
    return max(time1, time2)                     // imh = image height
}

double timeSBTscatter() {
    time1 ← 0.0
    time2 ← 0.0
    for (i=1; i.leq.logP - logX; i++)
        time1 ← time1 + T_send,cc(imw · imh/(2 · i))
        time2 ← time2 + T_full,cc(imw · imh/(2 · i))
    }
    for (i=logP-logX+1; i.leq.logP; i++)
        time1 ← time1 + T_send,nc(imw · imh/(2 · i))
        time2 ← time2 + T_full,nc(imw · imh/(2 · i))
    }
    return max(time1, time2)
}

double timeBorderExchange() {                    // bw = border width
    return (2 · T_full,nn(bw · imh) + 2 · T_full,cc((imw + 2 · bw) · bh))   // bh = border height
}
```

Fig. 6. P-3PC estimation of OFT & SBT scatter and border exchange.

operation. An estimation of the related SBT gather operation is obtained as before.

A well-known method to implement the Gaussian convolution operations 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 1D 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 (see Fig. 7). First, each processing unit sends a subset of its local partial image to the neighboring unit on its right side in the logical CPU grid (if such neighbor exists). When a unit 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 neighboring node on the left. As shown in Fig. 7, 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 `timeBorderExchange()` operation in Fig. 6 gives a worst-case P-3PC model for this routine.

5 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 four different message sizes, arbitrarily set at 1K, 50K, 100K, and 500K (all 4-byte values). In addition, 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 benchmarking tool of Fig. 1 can be user-defined as well; the sizes given here are used by default.

Clearly, there is a tradeoff between the number of benchmarking operations to be performed and the obtainable estimation accuracy. Still, the predefined set of only four 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 below, actual message sizes range from 192 bytes up to 8 MB.

To give full insight in the benchmarking process, Fig. 8 gives a simplified overview in pseudocode. 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.

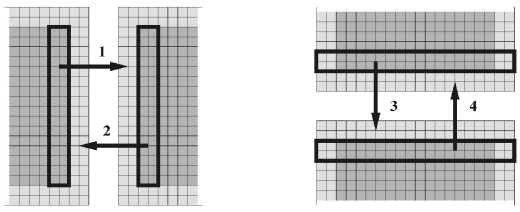

Fig. 7. Border exchange (right-left and down-up).

5.1 Distributed ASCI Supercomputer (DAS)

The first set of measurements was performed on the 128-node homogeneous DAS-cluster [3] located at the Vrije Universiteit

```
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·t_cf);
            }
        } else if (myCPU() .eq. 1)
            // matching send and recv calls at node 1 are not shown
    }
    return (total/nrRounds);
}
```

Fig. 8. Pseudocode for benchmarking all path-layout combinations. The constant time values t_{cs}, t_{cr}, and t_{cf} are obtained if `bufsize` equals zero.

in Amsterdam. All measurements were performed using MPI-LFC [5], 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 Table 1. 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.

TABLE 1
Benchmarking Results Obtained on DAS (in μs)

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

In the following, we show the results as obtained for the example application of Section 4. For each of the communication algorithms, we have been careful to keep the intrusiveness of the measurements to a minimum. All P-3PC estimations are obtained as in Fig. 6. 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 Table 1, we have reduced the P-3PC model into LogGP in the following manner: $g = t_{cs}$, $L = t_{cf}$, and $G = t_{af,cc}$. As indicated in Section 3.2, 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 Fig. 9a results are presented for a 512^2 floating point image, which is mapped onto a 1×16 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.

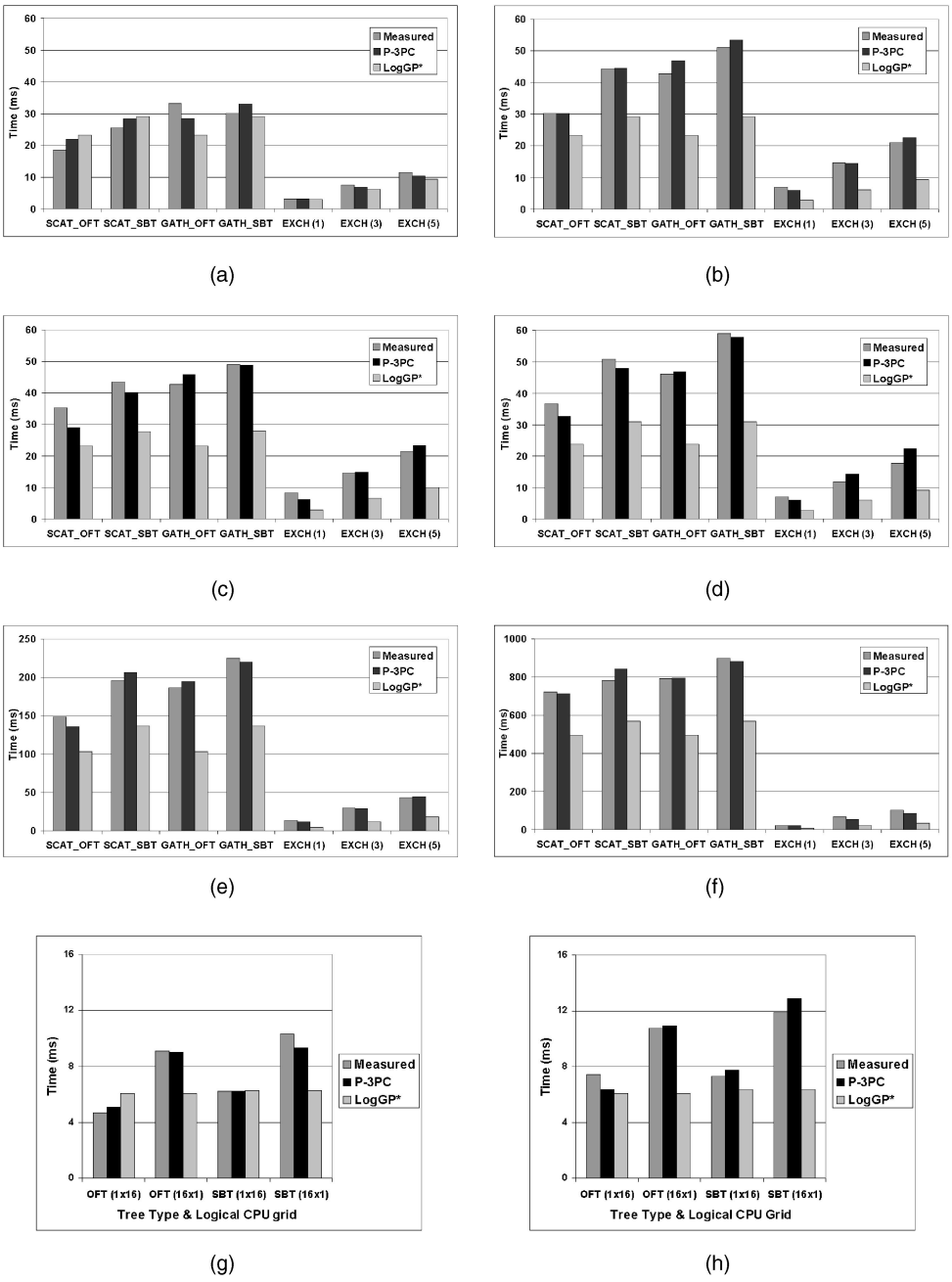

Fig. 9. Measurements (DAS) versus P-3PC & LogGP* estimations (1). (a) 512^2 image on 1×16 CPU grid. (b) 512^2 image on 16×1 CPU grid. (c) 512^2 image on 8×1 CPU grid. (d) 512^2 image on 32×1 CPU grid. (e) $1,024^2$ image on 16×1 CPU grid. (f) $2,048^2$ image on 16×1 CPU grid. (g) Scatter (256^2 image). (h) Gather (256^2 image).

As can be seen in Fig. 9b, for a 16×1 data decomposition, P-3PC outperforms LogGP* by far. This is because for such decomposition all messages involve noncontiguous data at the sender side. Figs. 9c and 9d show similar results for 8×1 and 32×1 decompositions. A comparison for larger image data structures is shown in Figs. 9e and 9f. 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 [20], [21].

Figs. 9g and 9h show that the P-3PC model indeed allows the scheduler of Section 2 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.

Fig. 10 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 Figs. 10a, 10b, 10c, and 10d that,

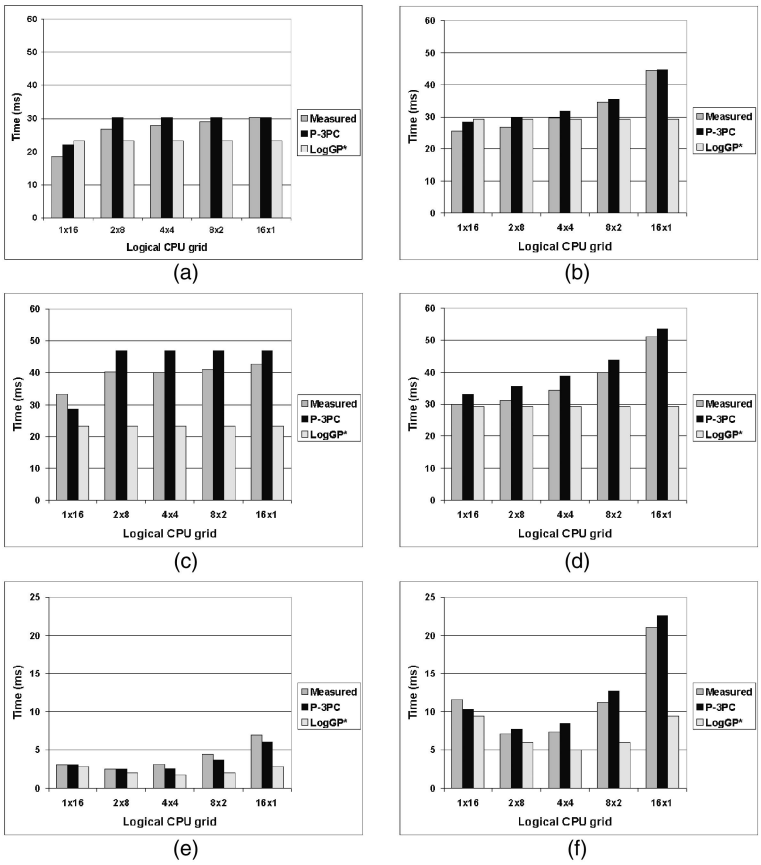

Fig. 10. Measurements (DAS) versus P-3PC & LogGP* estimations (2). (a) OFT Scatter image (512^2). (b) SBT Scatter image (512^2). (c) OFT Gather image (512^2). (d) SBT Gather image (512^2). (e) Exchange (512^2 image, $\sigma = 1$). (f) Exchange (512^2 image, $\sigma = 5$).

while for all but the 1×16 decomposition, P-3PC is somewhat pessimistic, the estimations get better for decompositions that are "closer" to 16×1. 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 16×1 decomposition (again, see also [20], [21]).

Figs. 10e and 10f 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 [10] for such application that even applies values of $\sigma > 5$), these results are important indeed. For additional results obtained on the DAS (also including sequential computation), we refer to [24].

5.2 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 [11]. 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 22.17.

Because the Beowulf cluster is heavily used for other research projects as well, we have been able to use only eight nodes at a time. Fig. 11 presents results for all algorithms, using a 512^2 floating point image which is mapped onto a 1×8 grid as well as a 8×1 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 nonunit-stride memory access is not caused by buffer copying, but can be attributed to caching alone. Although less significant on the Beowulf cluster, this is exactly the effect Prieto et al. have shown to be important on other parallel machines [20], [21].

6 Conclusions

In this paper, 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.

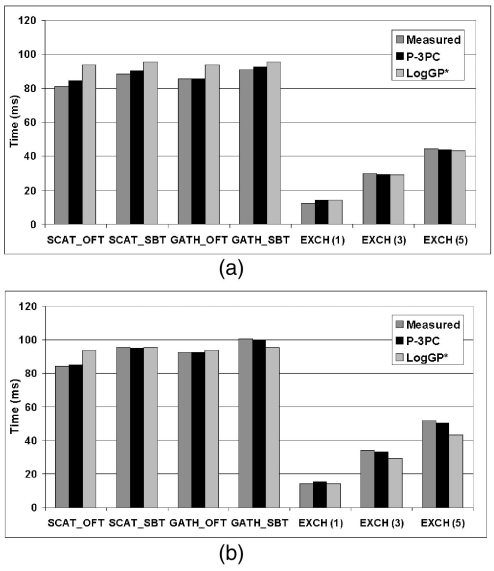

Fig. 11. Measurements (Beowulf) versus P-3PC and LogGP* estimations.

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 [7]). The model cannot 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 [2].

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 still can be captured due to the semiempirical modeling approach described in Section 3.3.1. 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 enough to obtain accurate predictions. The P-3PC model merely acknowledges that nonlinearities in communication costs may be significant (as shown in Section 2.1) 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 paper—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 [20], [21], we expect the model to be applicable to the very same machines and MPI implementations as well.

ACKNOWLEDGMENTS

The authors would like to thank Andrew Bagdanov, Zeger Hendrikse (University of Amsterdam), and Henk Sips (Delft University of Technology) for their comments on a preliminary version of this paper. The authors would also like to thank the anonymous reviewers for their thoughtful and constructive comments. This work was supported by NWO (Nederlandse Organisatie voor Wetenschappelijk Onderzoek) under grant 612-11-000.

REFERENCES

[1] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model—One Step Closer Towards a Realistic Model for Parallel Computation" *Proc. Symp. Parallel Algorithms and Architectures (SPAA),* pp. 95-105, July 1995.

[2] C. Andras Moritz and M.I. Frank, "LoGPC: Modeling Network Contention in Message-Passing Programs," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 4, pp. 404-415, Apr. 2001.

[3] H.E. Bal et al., "The Distributed ASCI Supercomputer Project," *Operating Systems Rev.,* vol. 34, no. 4, pp. 76-96, Oct. 2000.

[4] A. Bar-Noy and S. Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems," *Math. Systems Theory,* vol. 27, no. 5, pp. 431-452, 1994.

[5] R.A.F. Bhoedjang, T. Rühl, and H.E. Bal, "LFC: A Communication Substrate for Myrinet," *Proc. Conf. Advanced School for Computing and Imaging (ASCI '98),* pp. 31-37, June 1998.

[6] J. Bruck et al., "On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 3, pp. 256-265, Mar. 1996.

[7] D. Culler et al., "LogP: Towards a Realistic Model of Parallel Computation," *Proc. Fourth ACM SIGPLAN Symp. Principles and Practice of Parallel Programming,* pp. 1-12, May 1993.

[8] I.T. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison-Wesley, 1995.

[9] A. Geist et al., *PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[10] J.M. Geusebroek, A.W.M. Smeulders, and H. Geerts, "A Minimum Cost Approach for Segmenting Networks of Lines," *Int'l J. Computer Vision,* vol. 43, no. 2, pp. 99-111, July 2001.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing,* vol. 22, no. 6, pp. 789-828, Sept. 1996.

[12] S.E. Hambrusch and A. Khokhar, "C^3: A Parallel Model for Coarse-Grained Machines," *J. Parallel and Distruted Computing,* vol. 32, no. 2, pp. 139-154, 1996.

[13] Z. Juhasz, "An Analytical Method for Predicting the Performance of Parallel Image Processing Operations" *The J. Supercomputing,* vol. 12, nos. 1/2, pp. 157-174, 1998.

[14] M. Lauria, "LogP Characterization of FM on the VU's DAS Machine," technical report, Dipartimento di Informatica e Sistemistica, Univ. di Napoli Federico II, 1997.

[15] O.A. McBryan, "An Overview of Message Passing Environments," *Parallel Computing,* vol. 20, no. 4, pp. 417-444, Apr. 1994.
[16] W.F. McColl, "Scalability, Portability and Predictability: The BSP Approach to Parallel Programming," *Future Generation Computer Systems,* vol. 12, pp. 265-272, 1996.
[17] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard (version 1.1)," technical report, Univ. of Tennessee, Knoxville, Tenn., June 1995.
[18] Message Passing Interface Forum, "MPI-2: Extensions to the Message-Passing Interface," technical report, Univ. of Tennessee, Knoxville, Tenn., July 1997.
[19] N. Nupairoj and L. M. Ni, "Performance Evaluation of Some MPI Implementations on Workstation Clusters," *Proc. 1994 Scalable Parallel Libraries Conf. (SPLC' 94),* pp. 98-105, Oct. 1994.
[20] M. Prieto, I.M. Llorente, and F. Tirado, "A Review of Regular Domain Partitioning," *SIAM News,* vol. 33, no. 1, Jan. 2000.
[21] M. Prieto, I.M. Llorente, and F. Tirado, "Data Locality Exploitation in the Decomposition of Regular Domain Problems," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 11, pp. 1141-1149, Nov. 2000.
[22] F.J. Seinstra and D. Koelma, "Modeling Performance of Low Level Image Processing Routines on MIMD Computers," *Proc. Conf. Advanced School for Computing and Imaging, (ASCI '99),* pp. 307-314, June 1999.
[23] F.J. Seinstra and D. Koelma, "P-3PC: A Simple and Accurate Model of Point-to-Point Communication," technical report, ISIS, Faculty of Science, Univ. of Amsterdam, The Netherlands, Dec. 2000.
[24] F.J. Seinstra, D. Koelma, and J.M. Geusebroek, "A Software Architecture for User Transparent Parallel Image Processing on MIMD Computers," *Proc. Seventh Int'l Euro-Par Conf. (Euro-Par 2001),* pp. 653-662, Aug. 2001.
[25] A.J. van der Steen and R. van der Pas, "A Performance Analysis of the SGI Origin 2000," *Proc. Third Int'l Meeting Vector and Parallel Processing,* pp. 534-547, June 1998.
[26] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM,* vol. 33, no. 8, pp. 103-111, Aug. 1990.
[27] Z. Xu, X. Zhang, and L. Sun, "Semi-Empirical Multiprocessor Performance Predictions," *J. Parallel and Distributed Computing,* vol. 39, no. 1, pp. 14-28, 1996.
[28] X. Zhang, Y. Yan, and K. He, "Latency Metric: An Experimental Method for Measuring and Evaluating Parallel Program and Architecture Scalability," *J. Parallel and Distributed Computing,* vol. 22, no. 3, pp. 392-410, 1994.

Frank J. Seinstra received the MSc degree in computer science from the Vrije Universiteit in Amsterdam in 1996. He is currently finishing the PhD thesis (entitled: "User Transparent Parallel Image Processing") at the University of Amsterdam. His research interests include parallel and distributed programming, automatic parallelization, performance modeling, and scheduling, especially in the application area of image and video processing.

Dennis Koelma received the MSc and PhD degrees in computer science from the University of Amsterdam in 1989 and 1996, respectively. The subject of his thesis is "A Software Environment for Image Interpretation". Currently, he is working on Horus: a software architecture for doing research in accessing the content of digital images. His research interests include image and video processing, software architectures, parallel programming, databases, graphical user interfaces, and image information systems.