

Conf-9410337--1

CONTRIBUTED PAPER

Presented at the Scalable Parallel Libraries Conference, 1994
October 12-14, 1994

An MPI Version of the BLACS¹

David W. Walker
Mathematical Sciences Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6367
(615) 574-7401 (office)
(615) 574-0680 (fax)
walker@msr.epm.ornl.gov

The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-84OR21400. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

¹This work was supported in part by ARPA under contract number DAAL03-91-C-0047 administered by ARO, and in part by DOE under contract number DE-AC05-84OR21400. with the Martin Marietta Energy Systems, Inc.

MASTER

REPRODUCTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

An MPI Version of the BLACS[†]

D. W. Walker
Mathematical Sciences Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6367

Abstract

In this paper, issues related to implementing an MPI version of the Basic Linear Communication Subprograms (BLACS) are investigated. A set of routines, the MPI Linear Algebra Communication Subprograms (MLACS), are presented, and these are used to implement an MPI version of the BLACS. The MLACS provide the same functionality as the BLACS, but extend the functionality of the BLACS to include both blocking and nonblocking communication, and all four of the MPI communication modes.

1 Introduction

The Basic Linear Algebra Communication Programs (BLACS) are message passing routines that communicate matrices among processes arranged with a two-dimensional, logical, process topology [1, 2, 4]. In addition to point-to-point and broadcast communication routines, the BLACS also include simple reduction operations. Routines are also provided for communicating trapezoidal matrices, thereby avoiding the need to communicate unnecessary parts of triangular and symmetric matrices.

MPI is a proposed standard for message passing [3] which provides a wide variety of point-to-point and collective communication routines. Support is provided for *process groups*, so that a particular communication operation can be restricted to involve only a given set of processes. In MPI a process is identified by a group and its rank within that group. A process may belong to several groups. In point-to-point communication, messages are regarded as being labeled by a *communication context*, and a tag relative to that context. Communication contexts are a means within MPI of ensuring that messages intended for receipt in one phase of an application cannot be incorrectly received in another phase. Communication

contexts are managed by MPI and are not visible at the application level. Messages in MPI are typed, and *general datatypes* are supported. These may be used for communicating array sections and irregular data structures.

The next section gives a brief overview of the BLACS and points out some differences in the semantics and syntactic style of the MPI and BLACS routines. Section 3 gives an overview of the MPI Linear Algebra Communication Subprograms (MLACS), and their implementation is presented in Section 4. The MLACS provide the same functionality as the BLACS, but their syntax differs since they conform to an MPI style of interface. In Section 5, the implementation of the BLACS on top of the MLACS is discussed. It should be noted that the MPI implementation of the BLACS described in this section is experimental, and does not represent a definitive MPI implementation of the BLACS. Some concluding remarks are made in Section 6.

2 An Overview of the BLACS

In this section a brief overview of the BLACS will be given. The BLACS consist of point-to-point communication, collective communication, and auxiliary routines. Processes are referenced by their location in a two-dimensional logical process grid, or process topology. All BLACS communication routines are passed a context argument which uniquely identifies the logical process grid that the communication operation is performed on.

2.1 Point-To-Point Communication

We first consider point-to-point communication routines. The calling sequences of the BLACS point-to-point communication routines are as shown in Fig. 3 in the Appendix. The first letter of these routines, denoted in Fig. 3 by *v*, indicates to which datatype the routine pertains. Thus, if *v* is *I*, for

[†]This work was supported in part by ARPA under contract number DAAL03-91-C-0047 administered by ARO, and in part by DOE under contract number DE-AC05-84OR21400.

UPLO	$m - n < 0$	$m - n \geq 0$
'U'		
'L'		

Figure 1: The dependence of the shape of the trapezoid on the values of `uplo` and $m-n$.

example `IGESD2D`, the routine handles integer data. Other valid Fortran datatypes are real (`S`), double precision (`D`), complex (`C`), and double complex (`Z`), where the letter in parentheses shows what should be substituted for `v`. The routines `vGESD2D` and `vGERV2D` are for sending and receiving general rectangular matrices, respectively. The routines `vTRSD2D` and `vTRRV2D` are for sending and receiving trapezoidal matrices. In all the routines, `A` is the source or destination matrix, `LDA` is its leading dimension as specified in its declaration, and `CONTEXT` is the context. In the send routines `RDEST` and `CDEST` are the row and column in the process topology of the destination process. In the receive routines `RSRC` and `CSRC` are the row and column of the source process. For general matrices, `M` and `N` give the number of rows and columns, respectively, in the source or destination matrix.

The BLACS routines for communicating trapezoidal matrices have an argument `UPLO` which specifies an upper trapezoidal matrix if it has the value `'U'`, and a lower trapezoidal matrix if it has the value `'L'`. In both cases the actual trapezoid communicated depends on the value of $m-n$, as shown in Fig. 1. Another input argument, `DIAG`, specifies whether the trapezoid has ones on the diagonal. If `DIAG` has the value `'U'` then the diagonal is unity and the send and receive routines do not reference these elements, i.e., they are not communicated. If `DIAG` has the value `'N'` then the diagonal is not unity and these elements are communicated.

An important feature of the BLACS point-to-point communication routines is that messages have no tags. If the underlying communication layer permits message selectivity on tag, but not on source process, then

these tags are generated locally within the BLACS using a simple algorithm based on the process number of the sender and the previous communication between the two processes. This approach is used, for example, to implement the BLACS on top of Intel's NX communication system. MPI permits messages to be selected according to tag and source process, so in the MPI implementation of the BLACS described in Section 5 tags do not need to be generated.

The BLACS provide a mechanism for reserving a range of tag values for use with each BLACS context. Problems may arise if other components of an application, for example other library packages, are not aware of the portions of tag space reserved by the BLACS. For example, suppose one process sends data with a non-BLACS routine before entering a BLACS communication routine, and the destination process makes a matching non-BLACS receive call after exiting the BLACS routine. In this case the non-BLACS send may be incorrectly matched by a receive within the BLACS routine. Problems of indeterminacy may arise if a process enters a BLACS communication routine with unmatched receive operations still outstanding, since these may be incorrectly matched in the BLACS routine. In the MPI implementation of the BLACS described in Section 5, these problems with communication safety can be avoided in MPI programs.

2.2 Collective Communication

The BLACS provide routines for broadcasting general rectangular matrices and trapezoidal matrices, and routines for reducing matrices. All the BLACS collective communication routines, shown in Fig. 3, are passed an argument, `TOP`, specifying the network topology that should be emulated during the communication, and an argument, `SCOPE`, which specifies the processes involved in the communication operation. There are three possible values for `SCOPE`, depending on whether all the processes in the process topology are involved (`'A'`), or just the processes in a row (`'R'`) or column (`'C'`) of the process topology are involved.

The routines `vGEBS2D` and `vGEBR2D` are for sending and receiving the broadcast of a general rectangular matrix, respectively. For broadcasting trapezoidal matrices the routines `vTRBS2D` and `vTRBC2D` are used.

The routines `vGSUM2D`, `vGMAX2D`, and `vGMIN2D` are collective reduction routines that return their results to the root process at row `RDEST` and column `CDEST` of the process topology. If `RDEST` has the value -1 then the results are returned to all processes. Each process, p , is assumed to hold a matrix A^p , and an elementwise reduction is applied to these matrices. Thus, for the

routine `vGSUM2D`, if process r is the root and $a_{i,j}^p$ is the (i, j) th element of the matrix A^p on process p , then

$$a_{i,j}^r \leftarrow \sum a_{i,j}^p \quad (1)$$

where the sum is performed over all the processes specified by the `SCOPE` argument. The routines `vGMAX2D` and `vGMIN2D` not only find the elementwise maximum or minimum of conformal matrices on each process, but optionally also return auxiliary arrays giving the location in the process topology at which the maximum or minimum occurs. Thus, in addition to returning to the root a matrix of maximum or minimum values, these two routines return matrices of row and column indices. The (i, j) element of the matrix, `RA`, of row indices gives the row in the process topology of the process for which $a_{i,j}^p$ is a maximum or minimum. The matrix of column indices `CA` is similarly defined. The argument `RCFLAG` is the leading dimension of the arrays `RA` and `CA`, or -1 if the location is not required.

3 An Overview of the MLACS

The MLACS routines provide the same functionality as the BLACS routines, but have an MPI style of interface. In the MLACS, point-to-point messages are typed and tagged so that the corresponding routines have datatype and message tag arguments. The source process and message tag may be wildcarded in an MLACS receive call to indicate that any value will be accepted as a match. The source process is wildcarded by setting the row and column of the source process in the process topology to the named constant values `MLACS_ANY_ROW` and `MLACS_ANY_COLUMN`, respectively. It is not permitted to have just one of the row or column indices wildcarded — either both must be wildcarded, or neither.

In MPI, point-to-point messages may be blocking or nonblocking, whereas in the BLACS they are blocking. Furthermore, in MPI messages may be sent in one of four communication modes (standard, ready, synchronous, and buffered). The semantics of the BLACS send routines correspond to the standard mode. The MLACS have the same sorts of point-to-point communication routines as MPI, and provide separate routines for blocking and nonblocking communication, and for each of the four send communication modes. Thus, there are 8 MLACS routines for sending general matrices, 8 for sending trapezoidal matrices, 2 for receiving general matrices, and 2 for receiving trapezoidal matrices. The names of the MLACS routines for the point-to-point communication of gen-

SEND	Blocking	Nonblocking
Standard	<code>mlacs_send</code>	<code>mlacs_isend</code>
Ready	<code>mlacs_rsend</code>	<code>mlacs_irsend</code>
Synchronous	<code>mlacs_ssend</code>	<code>mlacs_issend</code>
Buffered	<code>mlacs_bsend</code>	<code>mlacs_ibsend</code>

RECEIVE	Blocking	Nonblocking
Standard	<code>mlacs_recv</code>	<code>mlacs_irecv</code>

Figure 2: Names of the MLACS point-to-point send and receive routines for general rectangular matrices.

eral rectangular matrices are given in Fig. 2. The routines for communicating trapezoidal matrices just have “_TRAP” appended. For example, the routine for performing a blocking receive of a trapezoidal matrix is `MLACS_RECV_TRAP`. The nonblocking send and receive routines are similar to the corresponding MPI routines. An identifier is returned which identifies the communication operation, and may be subsequently used to test for completion of the operation using the MLACS routines `MLACS_WAIT` and `MLACS_TEST`. These are analogous to the MPI routines `MPI_WAIT` and `MPI_TEST` which, respectively, block until completion of a communication operation, and return a flag giving the status.

All MLACS communication routines have a communicator argument. The MLACS collective communication routines have a datatype argument, but like MPI have no message tag. In the MLACS a single routine is used for broadcasting data from a root process and receiving the data on all the other processes. For general matrices the routine `MLACS_BCAST` is used, and for trapezoidal matrices the routine `MLACS_BCAST_TRAP`. The MLACS routines for reducing a matrix of values over a set of processes are called `MLACS_REDUCE_SUM`, `MLACS_REDUCE_MAXLOC`, and `MLACS_REDUCE_MINLOC`. In the MLACS, only the sum reduction operation is defined for complex data types as MPI does not define minimum and maximum reduction functions for complex data types. The arguments to `MLACS_REDUCE_MAXLOC` and `MLACS_REDUCE_MINLOC` differ in an important way from those of their BLACS counterparts. Instead of returning matrices of row and column indices to give the location of the maximum or minimum in the process topology, the MLACS routines just return a matrix of process ranks relative to the communicator used in the call. The reasons for this difference are explained in the next section.

4 Implementation of the MLACS

In this section an implementation of the MLACS will be presented. The communication routines usually consist of the following stages,

1. check initialization, validate input parameters;
2. translate between process coordinates and rank;
3. create general datatype for the communication;
4. call MPI routine to communicate data;
5. free the general datatype.

Here we shall just be concerned with validating input arguments that are specific to MPI, i.e., the communicator argument passed to all MLACS communication routines. In a full implementation the non-MPI input arguments should also be validated. Validating the communicator involves checking that it has a one- or two-dimensional Cartesian topology. As shown in Fig. 5 in the Appendix, the type of topology associated with the communicator is determined with a call to `MPI_TOPO_STATUS`, and the dimension of the topology is given by `MPI_CARTDIM_GET`. The routine `MLACS_ABORT` is called if the topology is invalid. This routine is passed an error code which is returned to the invoking environment. The meaning of the error code is dependent on the MPI implementation. In Fig. 5 and subsequent figures the error code passed to `MLACS_ABORT` is denoted by "`<error>`" to indicate that the appropriate implementation-dependent error code should be substituted.

Translating from process coordinates to rank is done by calling the routine `MPI_CART_RANK`, and the inverse translation from rank to process coordinates is done by calling `MPI_CART_COORDS`.

The example implementations given in this section do not check the error code returned by the MPI routines, nor do they attempt to return a valid error code. In a full implementation these issues would have to be addressed.

4.1 Implementation of Point-to-point Routines

Figure 5 shows an implementation of the routine `MLACS_SEND`. The routine `INITIALIZE_SEND` calls `MLACS_INITIALIZED` to check that the MLACS have been previously initialized by a call to `MLACS_GRIDINIT` or `MLACS_GRIDMAP`. Next `INITIALIZE_SEND` validates the topology, translates the coordinates of the destination process to a rank, and then

returns. In `MLACS_SEND` a general datatype is constructed that refers to a rectangular $m \times n$ submatrix. This submatrix consists of n blocks of m consecutive elements (the matrix columns) separated by the leading dimension of the matrix, `lda`. The routine `MPI_TYPE_VECTOR` is used to create the general datatype, `mtype`. The call to `MPI_TYPE_COMMIT` tells MPI that `mtype` is going to be used in a communication operation, rather than being an intermediate stage in the construction of a more complex datatype. The data are actually sent by the call to `MPI_SEND` which performs a blocking send in standard mode. Finally, the resources used by the datatype `mtype` are released by a call to `MPI_TYPE_FREE`.

In Fig. 6 we present an example implementation of the routine `MLACS_RECV`. The structure of this routine is similar to that of `MLACS_SEND`, except for the translation between process coordinates and rank in the routine `INITIALIZE_RECV`. Here added complication arises from the need to handle the case in which the source process is wildcarded. It is not possible to wildcard just the row index or just the column index of the source process. Either both, or neither, of the row and column indices must be wildcarded. The MLACS named constants `MLACS_ANY_ROW` and `MLACS_ANY_COLUMN` are used for wildcarding the row and column indices, respectively. The data are actually received by the MPI routine `MPI_RECV`. The MPI return status, `rstatus`, is returned by `MLACS_RECV`, and may be used to determine the actual source process and tag if either or both were wildcarded.

In Fig. 7 we present an implementation of the MLACS routine `MLACS_SEND_TRAP` for sending a trapezoidal matrix from one process to another. In this routine a general datatype, `mtype`, must be constructed that corresponds to the appropriate trapezoidal matrix, and this makes the code rather more complicated than for the case of general rectangular matrices. In the trapezoidal case each column of the matrix to be communicated is now of a different length. Also if `uplo` is 'L' then the stride between successive columns is not constant. We, therefore, use the routine `MPI_TYPE_INDEXED` to construct the general datatype, since this routine can handle variable block sizes and strides. The routine is passed an array `work` of size at least $2m$ which is partitioned into two parts. The lower part of the array is used to store the number of elements in each column of the trapezoidal matrix. The upper part of the `work` array is used to store the offset in elements of the start of each column, relative to the start of the first column. This informa-

tion is evaluated in the routine `SETUP_INDEXED` and is then passed into the routine `MPI_TYPE_INDEXED` which treats each column as a separate block, and needs to know the length and starting offset of each block. In `SETUP_INDEXED` the arrays for the `uplo = 'U'` case are first set up, and this information is then used to deduce the arrays for the `uplo = 'L'` case, if necessary. An implementation for the routine `MLACS_RECV_TRAP` is shown in Fig. 9.

4.2 Implementation of Collective Routines

In this section, implementations of the MLACS broadcast and reduction routines which comprise the MLACS collective communication routines will be considered. As discussed in Section 2.2, the BLACS use separate routines to send and receive broadcast data. In the MLACS a single broadcast routine is provided for each matrix type (general and trapezoidal). This approach is consistent with the MPI specification. The routine `MLACS_BCAST` for broadcasting a general rectangular matrix is simple to implement, and an implementation is given in Fig. 8. Note that if a one-dimensional process topology is used to broadcast over just a row or column of processes, then either the argument `rroot` or `croot` is unnecessary and must be replaced by the named constant `MLACS_IGNORE`.

The routine `MLACS_BCAST_TRAP` for broadcasting a trapezoidal matrix is rather more complicated than that for general matrices. As with the point-to-point communication of trapezoidal matrices, the extra complication comes from constructing the general datatype for specifying a trapezoidal matrix. The code for doing this is the same as in the point-to-point routines. An implementation of the MLACS routine `MLACS_BCAST_TRAP` is given in Fig. 10

In Fig. 11 an implementation of the routine `MLACS_REDUCE_SUM` is shown. This routine does an elementwise sum of a matrix. The routine `INITIALIZE_COLLECTIVE` is called to check that the MLACS have been initialized, and that the input communicator has the correct type of topology. `INITIALIZE_COLLECTIVE` also translates the process coordinates of the root, `rroot` and `croot`, to a process rank. Then each matrix column is reduced in turn by applying the MPI routine `MPI_REDUCE`. A faster implementation would copy the input matrix into contiguous storage and then apply `MPI_REDUCE` to all the data in a single call, but this would require more working storage.

The routines `MLACS_REDUCE_MIN` and `MLACS_REDUCE_MAX` not only find the element-

wise minimum of conformal matrices on each process, but also return auxiliary arrays giving the location in the process topology at which the minimum or maximum occurs. An important difference between the BLACS and the MLACS is that in the BLACS the routines `vGMIN2D` and `vGMAX2D` return two matrices giving the row and column indices of the location in the process topology of the process containing the minimum or maximum value. However, the corresponding routines in the MLACS, `MLACS_REDUCE_MIN` and `MLACS_REDUCE_MAX`, return a matrix of process ranks to indicate the location of the process containing the minimum or maximum value. These ranks are relative to the group associated with the communicator input to the routine. The MLACS routines were designed in this way because in the BLACS the row and column index matrices contain process ranks relative to the full 2D process topology. However, if a row- or column-oriented communicator is passed into one of the MLACS routines there is no way for the routine to compute a location in the full 2D process topology (unless the 2D communicator is also passed into the routine). The row and column information can easily be recovered upon return from `MLACS_REDUCE_MINLOC` or `MLACS_REDUCE_MAXLOC` by running the matrix of ranks through the routine `MPI_CART_COORDS`.

In Fig. 12, `a` is the input matrix to be reduced, and `b` is the output matrix of minimum values. The array `ranks` is the matrix giving the rank of the process for which the corresponding value in `a` is a minimum.

Next a general datatype, `mtype`, corresponding to a pair of variables of type `datatype` is created with a call to `MPI_TYPE_CONTIGUOUS`. The routine then reduces each column of the matrix `a` in turn. For each column the value of `a` and the process rank `myrank` are copied into the array `work`. Thus for the i th column, `work` contains $(a_{1i}, \text{myrank}), (a_{2i}, \text{myrank}), \dots, (a_{mi}, \text{myrank})$, which we can treat as a vector of length m of type `mtype`. The array `work` must be declared in the calling subprogram to have the same datatype as the matrix `a` (i.e., type `datatype`), and to be at least $4m$ in length. The routine `MPI_REDUCE` is called to perform the reduction using the pre-defined MPI function `MPL_MINLOC`. The results are returned into the upper half of the `work` array, and these are then unpacked into the matrices `b` and `ranks`, which are returned to the calling subprogram. The routine `MLACS_REDUCE_MAXLOC` is implemented in a very similar way.

The above routines leave the results on just the root process. The MLACS include a similar set of routines (`MLACS_ALLREDUCE_SUM`, `MLACS_ALLREDUCE_MIN`

LOC, and MLACS_ALLREDUCE_MAXLOC) that leave the results on all processes. Their implementation follows that described above, except the MPI routine MPI_ALLREDUCE is called instead of MPI_REDUCE.

4.3 Support Routines

Only some of the BLACS support routines have counterparts in the MLACS. An MLACS analog, MLACS_GRIDINIT to the BLACS routine BLACS_GRIDINIT is provided. This establishes communicators for the one- and two-dimensional process topologies used in the MLACS. The process groups associated with these three output communicators correspond to all the processes, and the row and column of processes of which the calling process is a member. In a simple implementation, MLACS_GRIDINIT is a convenience function - all it does is return communicators for use in MLACS routines. In more sophisticated implementations it may also preallocate resources to be used by the MLACS routines. The MLACS also include a routine MLACS_INITIALIZED which returns a flag to indicate if the MLACS have previously been initialized by a call to MLACS_GRIDINIT or MLACS_GRIDMAP. The MLACS routine MLACS_GRIDEXIT should be called when all MLACS calls for a particular grid have been completed, and releases resources used in the MLACS routines, such as the three communicators created by calling MLACS_GRIDINIT or MLACS_GRIDMAP. Implementations of the routines MLACS_GRIDINIT and MLACS_GRIDEXIT are given in Fig. 4.

The MLACS counterparts of the BLACS routines BLACS_PINFO, BLACS_GRIDINFO, BLACS_PNUM, BLACS_PCOORD, and BLACS_BARRIER are MLACS_PROCINFO, MLACS_GRIDINFO, MLACS_CART_RANK, MLACS_CART_COORDS, and MLACS_BARRIER. These routines are simple to implement using MPI so we will not give the code here. It might be argued that there is little point in having MLACS routines, such as MLACS_BARRIER, consisting of just one call to an MPI routine. However, we believe this approach is useful since it allows in most cases just MLACS routines to be used between a call to MLACS_GRIDINIT and the matching MLACS_GRIDEXIT.

The BLACS routine BLACS_GRIDMAP can be used to form a grid from a set of processes, and allows the placement of processes on the grid to be controlled. The MLACS provide an analogous routine, MLACS_GRIDMAP, shown in Fig. 13. This routine creates a grid of `nprows` rows and `npcols` columns. The argument `usermap` is a matrix with leading dimension

`ldu` whose (i, j) th entry is the rank of the process at location (i, j) in the grid. `comm` is a communicator out of whose group the grid is formed. MLACS_GRIDMAP returns the communicators `commall`, `commrow`, and `commcol` of the new grid.

The routine MLACS_GRIDMAP first finds the rank of the calling process using MPI_COMM_RANK. For each process in the grid the rank, `newrank`, in the grid is found, and the value of `color` is set to zero. For the other processes `color` is set to MPI_UNDEFINED. Here we make use of the fact that both MPI and the BLACS arrange the processes of a two-dimensional topology by increasing rank in row-major order. The routine MPI_COMM_SPLIT is called to create the communicator, `newcomm`, whose group consists of just the processes in the grid. For those processes in the grid the communicator `newcomm` is then passed to the routine MLACS_GRIDINIT. This creates a grid communicator, `commall`, with a two-dimensional topology, and the row and column communicators, `commrow` and `commcol`, for the grid. For processes outside the grid `commall`, `commrow`, and `commcol` are all returned with the value MPI_COMM_NULL.

Most of the other BLACS auxiliary routines are specific to PVM, or pertain to the generation of unique tags, and so are not needed in the MLACS. The routine SETBRANCHES can be discarded because the MLACS do not specify how communication is to be performed. Finally, FREEBUFF can also be discarded as MPI will manage message buffers.

5 Implementing the BLACS on top of the MLACS

It is quite simple to implement the BLACS using the MLACS. To do this the BLACS routines need to know the communicator for all the processes in the two-dimensional topology, and the communicators for the processes in a row and column of the topology. We make use of MPI's caching facility to cache the 2D grid, row and column communicators, returned by a call to MLACS_GRIDINIT or MLACS_GRIDMAP, with the communicator input to these routines. Then the context argument in the BLACS routines is replaced by this communicator in the MPI implementation. This is a transparent change at the application level as BLACS contexts and MPI communicators are both integers in Fortran implementations. An implementation of the BLACS routine BLACS_GRIDINIT is shown in Fig. 15. First MLACS_GRIDINIT is called to create the three communicators `commall`, `commrow`, and `commcol`. Then each is cached with the com-

communicator `comm`. It is assumed that the keys `akey`, `rkey`, and `ckey` are initialized to the predefined value `MPI_KEYVAL_INVALID` in a block data routine.

The BLACS point-to-point routines can be implemented simply by recovering the communicator `commall` from the cache of communicator `comm`, and then calling one of the MLACS routines with the appropriate `datatype` argument. As an example, a simple implementation of the routine `IGESD2D` is given in Fig. 16.

In the implementation of the BLACS broadcast routines the appropriate communicator is recovered from the cache of the input communicator by examining the value of the `SCOPE` argument. A simple implementation of the BLACS routine `IGESB2D` is given in Fig. 17. Note that the `TOP` argument is ignored in this example. A full implementation of the BLACS would have to provide the different communication algorithms designated by the `TOP` argument.

The BLACS reduction routines `VGSUM2D` can be implemented in a very similar way to the BLACS broadcast routines, except in this case we call the MLACS routine `MLACS_REDUCE_SUM` or `MLACS_ALLREDUCE_SUM`. The implementation of the routines `VGMAX2D` and `VGMIN2D` is a little more complicated as in these routines the rank matrix returned by `MLACS_REDUCE_MAXLOC` or `MLACS_REDUCE_MINLOC` has to be converted into matrices of row and column indices. A simple implementation of `IGMIN2D` is shown in Fig. 18. It is necessary for `VGMIN2D` and `VGMAX2D` to pass a work array to `MLACS_REDUCE_MINLOC` and `MLACS_REDUCE_MAXLOC`. This work space could come from buffer space allocated when `BLACS_GRIDINIT` was called, or it could be created dynamically within the routine. In Fig. 18, we have not shown where the work space comes from. Similar issues arise in the communication of trapezoidal matrices.

The BLACS routine `BLACS_GRIDMAP` can be implemented using `MLACS_GRIDMAP`. The implementation is similar to that of the routine `BLACS_GRIDINIT`, shown in Fig. 15, except `MLACS_GRIDMAP` is called instead of `MLACS_GRIDINIT`, and the grid communicators are only cached if the calling process is in the grid. An example implementation is shown in Fig. 14.

6 Concluding Remarks

In this paper we have presented the MLACS, a set of routines for communicating matrices among processes arranged with a two-dimensional process topology. The MLACS are based on the MPI communica-

tion system, and reproduce nearly all of the functionality of the BLACS. The MLACS also extend the functionality of the BLACS by including blocking and non-blocking point-to-point communication routines, and the four MPI send communication modes. In addition, the MLACS permit the source process and message tag message selection criteria to be wildcarded by point-to-point receive routines. We have also shown how the MLACS can be used to develop a simple implementation of the BLACS. As well as serving as a basis for an MPI implementation of the BLACS, the MLACS can also be used directly in applications. Their use ensures communication safety, and correct behavior in multithreaded environments.

Acknowledgments

It is a pleasure to acknowledge the many helpful suggestions of Jack J. Dongarra and R. Clint Whaley.

References

- [1] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. van de Geijn. Basic Linear Algebra Communication Subprograms. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 287-290. IEEE Computer Society Press, 1991.
- [2] J. J. Dongarra, R. A. van de Geijn, and R. C. Whaley. A users' guide to the BLACS. Computer Science Dept. Technical Report, University of Tennessee, Knoxville, TN, December 1993.
- [3] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, April 1994. (To appear in the *International Journal of Supercomputer Applications*, Volume 8, Number 3/4, 1994).
- [4] R. C. Whaley. Basic linear algebra communication subprograms: Analysis and implementation across multiple parallel architectures. LAPACK Working Note 73, University of Tennessee, Knoxville, TN, 1994.

Appendix: Fortran Implementation

vGESD2D (CONTEXT,	M, N, A, LDA, RDEST, CDEST)
vGERV2D (CONTEXT,	M, N, A, LDA, RSRC, CSRC)
vTRSD2D (CONTEXT, UPLO, DIAG, M, N, A, LDA, RDEST, CDEST)	
vTRRV2D (CONTEXT, UPLO, DIAG, M, N, A, LDA, RSRC, CSRC)	
vGEBS2D (CONTEXT, SCOPE, TOP,	M, N, A, LDA)
vGEBR2D (CONTEXT, SCOPE, TOP,	M, N, A, LDA, RSRC, CSRC)
vTRBS2D (CONTEXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA)	
vTRBR2D (CONTEXT, SCOPE, TOP, UPLO, DIAG, M, N, A, LDA, RSRC, CSRC)	
vGSUM2D (CONTEXT, SCOPE, TOP, M, N, A, LDA,	RDEST, CDEST)
vGMAX2D (CONTEXT, SCOPE, TOP, M, N, A, LDA, RA, CA, RCFLAG, RDEST, CDEST)	
vGMIN2D (CONTEXT, SCOPE, TOP, M, N, A, LDA, RA, CA, RCFLAG, RDEST, CDEST)	

Figure 3: Calling sequences of the BLACS point-to-point and collective communication routines.

```

subroutine mlacs_gridinit (comm, nrow, ncol, commall, commrow, commcol, error)
integer comm, commall, commrow, commcol
integer nrow, ncol, error
integer dims(2), ndims
logical periods(2), reorder, remaindims(2)
data ndims, reorder, periods(1), periods(2) /2, 3*.false./

dims(1) = nrow
dims(2) = ncol
call mpi_cart_create (comm, ndims, dims, periods, reorder, commall, ierr)

remaindims(1) = .false.
remaindims(2) = .true.
call mpi_cart_sub (commall, remaindims, commrow, ierr)

remaindims(1) = .true.
remaindims(2) = .false.
call mpi_cart_sub (commall, remaindims, commcol, ierr)

return
end

subroutine mlacs_gridexit (commall, commrow, commcol, error)
integer comm, commall, commrow, commcol, error
integer ierr

call mpi_comm_free (commall, ierr)
call mpi_comm_free (commrow, ierr)
call mpi_comm_free (commcol, ierr)

return
end

```

Figure 4: Example Fortran 77 implementation of the MPI versions of MLACS_GRIDINIT and MLACS_GRIDEXIT.

```

subroutine mlacs_send (a, lda, m, n, datatype, rdest, cdest, tag, comm, error)
integer a(lda,*)
integer lda, m, n, rdest, cdest, error
integer datatype, tag, comm
integer ierr, rank
integer mtype

call initialize_send (comm, rdest, cdest, rank, ierr)

call mpi_type_vector (n, m, lda, datatype, mtype, ierr)
call mpi_type_commit (mtype, ierr)

call mpi_send (a, 1, mtype, rank, tag, comm, ierr)

call mpi_type_free (mtype, ierr)

return
end

subroutine initialize_send (comm, row, col, rank, error)
integer comm, row, col, rank, error
integer coords(2), ierr, status, ndims
logical flag

call mlacs_initialized (flag, ierr)
if (.not.flag) call mlacs_abort (comm, <error>, ierr)

call mpi_topo_test (comm, status, ierr)
if (status .ne. MPI_CART) call mlacs_abort (comm, <error>, ierr)

call mpi_cartdim_get (comm, ndims, ierr)
if (ndims .ne. 2) call mlacs_abort (comm, <error>, ierr)

coords(1) = row
coords(2) = col
call mpi_cart_rank (comm, coords, rank, ierr)

return
end

```

Figure 5: Fortran 77 code for the routine MLACS_SEND for sending a submatrix consisting of m rows and n columns of matrix a from the calling process to process at row $rdest$ and column $cdest$ of the two-dimensional process topology. This routine is blocking, and obeys the standard communication mode semantics of MPI.

```

subroutine mlacs_recv (a, lda, m, n, datatype, rsrc, csrc, tag, comm, rstatus, error)
integer a(lda,*)
integer lda, m, n, rsrc, csrc, rstatus, error
integer datatype, tag, comm
integer rank, mtype, ierr

call initialize_recv (comm, rsrc, csrc, rank, ierr)

call mpi_type_vector (n, m, lda, datatype, mtype, ierr)
call mpi_type_commit (mtype, ierr)

call mpi_recv (a, 1, mtype, rank, tag, comm, rstatus, ierr)
call mpi_type_free (mtype, ierr)

return
end

subroutine initialize_recv (comm, row, col, rank, error)
integer comm, row, col, rank, error
integer coords(2), ierr, status, ndims
logical flag

call mlacs_initialized (flag, ierr)
if (.not.flag) call mlacs_abort (comm, <error>, ierr)

call mpi_topo_test (comm, status, ierr)
if (status .ne. MPI_CART) call mlacs_abort (comm, <error>, ierr)

if (row .ne. MLACS_ANY_ROW) then
  if (col .ne. MLACS_ANY_COLUMN) then
    coords(1) = row
    coords(2) = col
    call mpi_cart_rank (comm, coords, rank, ierr)
  else
    [ an error occurred ]
    return
  endif
else
  if (col .eq. MLACS_ANY_COLUMN) then
    rank = MPI_ANY_SOURCE
  else
    [ an error occurred ]
    return
  endif
endif

return
end

```

Figure 6: Fortran 77 code for the routine MLACS_RECV for receiving a submatrix consisting of up to m rows and n columns of matrix a from the process at row $rsrc$ and column $csrc$ of the two-dimensional process topology. This routine is blocking.

```

subroutine mlacs_send_trap (a, lda, m, n, uplo, diag, datatype,
                           rdest, cdest, tag, comm, iwork, work, error)
integer a(lda,*), work(*)
integer lda, m, n, uplo, diag, rdest, cdest, iwork, error
integer datatype, tag, comm
integer rank, ierr, woff, mtype

call initialize_send (comm, rdest, cdest, rank, ierr)

woff = iwork/2
call setup_indexed (lda, m, n, uplo, diag, woff, work, ierr)
call mpi_type_indexed (n, work, work(woff), datatype, mtype)
call mpi_type_commit (mtype, ierr)

call mpi_send (a, 1, mtype, rank, tag, comm, ierr)

call mpi_type_free (mtype, ierr)

return
end

subroutine setup_indexed (lda, m, n, uplo, diag, woff, work, ierr)
integer lda, m, n, woff, work(*), ierr
character*1 uplo, diag
integer diagadj, col1, i, temp

diagadj = 0
if (diag .eq. 'U') diagadj = 1
col1 = max (0, m-n)
do i=0,n-1
  work(i+1) = min (col1+i+1,m) - diagadj
  work(i+woff) = i*lda
end do
if (uplo .eq. 'L') then
  do i=0,n/2
    temp = work(i+1)
    work(i+1) = work(n-i)
    work(n-i) = temp
  end do
  do i=0,n-1
    work(i+woff) = i*lda + m - work(i+1)
  end do
end if

return
end

```

Figure 7: Fortran 77 code for the routine MLACS_SEND_TRAP for sending a trapezoidal submatrix.

```

subroutine mlacs_bcast (a, lda, m, n, datatype, rroot, croot, comm, error)
integer a(lda,*)
integer lda, m, n, rroot, croot, error
integer datatype, comm
integer rank, mtype, root, ierr

call initialize_collective (comm, rroot, croot, rank, ierr)
call mpi_type_vector (n, m, lda, datatype, mtype, ierr)
call mpi_type_commit (mtype, ierr)
call mpi_bcast (a, 1, mtype, root, comm, ierr)
call mpi_type_free (mtype, ierr)

return
end

subroutine initialize_collective (comm, rroot, croot, rank, error)
integer comm, rroot, croot, rank, error
integer coords(2), ndims, status, ierr
logical flag

call mlacs_initialized (flag, ierr)
if (.not.flag) call mlacs_abort (comm, <error>, ierr)

call mpi_topo_test (comm, status, ierr)
if (status .ne. MPI_CART) call mlacs_abort (comm, <error>, ierr)

call mpi_cartdim_get (comm, ndims, ierr)
if (ndims .eq. 1) then
  coords(1) = rroot
  if (rroot .eq. MLACS_IGNORE) coords(1) = croot
else if (ndims .eq. 2) then
  coords(1) = rroot
  coords(2) = croot
else
  call mlacs_abort (comm, <error>, ierr)
end if
call mpi_cart_rank (comm, coords, rank, ierr)

return
end

```

Figure 8: Fortran 77 code for the routine MLACS_BCAST for broadcasting m rows and n columns of matrix a from the process at row $rroot$ and column $croot$ of the two-dimensional process topology.

```

subroutine mlacs_recv_trap (a, lda, m, n, uplo, diag, datatype,
                           rsrc, csrc, tag, comm, rstatus, iwork, work, error)
integer a(lda,*)
integer lda, m, n, uplo, diag, rsrc, csrc, error
integer datatype, tag, comm, rstatus, iwork, work(*)
integer rank, woff, mtype, ierr

call initialize_recv (comm, rsrc csrc, rank, ierr)

woff = iwork/2
call setup_indexed (lda, m, n, uplo, diag, woff, work, ierr)
call mpi_type_indexed (n, work, work(woff), datatype, mtype)
call mpi_type_commit (mtype, ierr)

call mpi_recv (a, 1, mtype, rank, tag, comm, rstatus, ierr)

call mpi_type_free (mtype, ierr)

return
end

```

Figure 9: Fortran 77 code for the routine MLACS_RECV_TRAP for receiving a trapezoidal submatrix.

```

subroutine mlacs_bcast_trap (a, lda, m, n, uplo, diag, datatype,
                             rroot, croot, tag, comm, iwork, work, error)
integer a(lda,*), work(*)
integer lda, m, n, uplo, diag, rroot, croot, error
integer datatype, tag, comm, iwork
integer rank, mtype, woff, ierr

call initialize_collective (comm, rroot, croot, rank, ierr)

call setup_indexed (lda, m, n, uplo, diag, woff, work, ierr)
call mpi_type_indexed (n, work, work(woff), datatype, mtype)
call mpi_type_commit (mtype, ierr)

call mpi_bcast (a, 1, mtype, rank, comm, ierr)

call mpi_type_free (mtype, ierr)

return
end

```

Figure 10: Fortran 77 code for the routine MLACS_BCAST_TRAP for broadcasting a trapezoidal submatrix.

```

subroutine mlacs_reduce_sum (a, b, ldab, m, n, datatype, rroot, croot, comm, error)
integer a(ldab,*), b(ldab,*)
integer ldab, m, n, rroot, croot, error
integer datatype, comm
integer root, ierr

call initialize_collective (comm, rroot, croot, root, ierr)

do i=1,n
  call mpi_reduce (a(1,i), b(1,i), m, datatype, MPI_SUM, root, comm, error)
end do

return
end

```

Figure 11: Fortran 77 code for the routine MLACS_REDUCE_SUM for the elementwise summation of general, rectangular matrices.

```

subroutine mlacs_reduce_minloc (a, b, ldab, m, n, ranks, ldia,
                               datatype, rroot, croot, comm, iwork, work, error)
integer a(ldab,*), b(ldab,*), ranks(ldia,*), work(*)
integer ldab, ldia, m, n, rroot, croot, iwork, error
integer datatype, comm
integer i, j, root, myrank, mtype, root, ierr

call initialize_collective (comm, rroot, croot, root, ierr)

call mpi_comm_type_contiguous (2, datatype, mtype)

call mpi_comm_rank (comm, myrank, ierr)

do i=1,n
  do j=1,m
    work(2*j-1) = a(j,i)
    work(2*j) = myrank
  end do

  call mpi_reduce (work, work(2*m+1), m, mtype, MPI_MINLOC, root, comm, error)
  do j=1,m
    b(j,i) = work(2*(j+m)-1)
    ranks(j,i) = int (work(2*(j+m)))
  end do
end do

return
end

```

Figure 12: Fortran 77 code for the routine MLACS_REDUCE_MINLOC for locating the elementwise minimum values of a matrix a, and the ranks of the processes containing the minimum values.

```

subroutine mlacs_gridmap (usermap, ldu, nrow, ncol, comm, commall, commrow, commcol, error)
integer usermap(ldu,*)
integer ldu, nrow, ncol, error
integer comm, commall, commrow, commcol
integer i, j, color, myrank, newrank, ierr

call mpi_comm_rank (comm, myrank, ierr)
color = MPI_UNDEFINED
do i=1,ncol
  do j=1,nrow
    if (myrank .eq. usermap(j,i)) then
      newrank = j-1+(i-1)*ncol
      color = 0
    end if
  end do
end do

call mpi_comm_split (comm, color, newrank, newcomm, ierr)

if (color .eq. 0) then
  call mlacs_gridinit (newcomm, nrow, ncol, commall, commrow, commcol, ierr)
else
  commall = MPI_COMM_NULL
  commrow = MPI_COMM_NULL
  commcol = MPI_COMM_NULL
end if

return
end

```

Figure 13: Fortran 77 implementation of the MLACS routine MLACS_GRIDMAP.

```

subroutine blacs_gridmap (comm, usermap, ldu, nrow, ncol)
integer comm
integer usermap(ldu,*), ldu, nrow, ncol
integer ierr, result, commall, commrow, commcol
integer akey, rkey, ckey
common /blacom/ akey, rkey, ckey
save /blacom/
:
:
call mlacs_gridmap (comm, usermap, ldu, nrow, ncol, commall, commrow, commcol, ierr)

call mpi_comm_compare (commall, MPI_COMM_NULL, result, ierr)
if (result .ne. MPI_IDENT) then
  call cache_communicator (comm, commall, akey, ierr)
  call cache_communicator (comm, commrow, rkey, ierr)
  call cache_communicator (comm, commcol, ckey, ierr)
end if
:
:
return
end

```

Figure 14: Implementation of the BLACS routine GRIDMAP.

```

subroutine blacs_gridinit (comm, nrow, ncol)
integer comm, nrow, ncol
integer commall, commrow, commcol, ierr
integer akey, rkey, ckey
common /blacom/ akey, rkey, ckey
save /blacom/

call mlacs_gridinit (comm, nrow, ncol, commall, commrow, commcol, ierr)

call cache_communicator (comm, commall, akey, ierr)
call cache_communicator (comm, commrow, rkey, ierr)
call cache_communicator (comm, commcol, ckey, ierr)

return
end

subroutine cache_communicator (comm, cachecomm, key, error)
integer comm, cachecomm, key, error
integer ierr, extra, dummy
logical useflag

if (key .eq. MPI_KEYVAL_INVALID)
  call mpi_keyval_create (MPI_NULLFN, MPI_NULLFN, key, extra, ierr)
call mpi_attr_get (comm, key, dummy, useflag, ierr)
if (.not.useflag) then
  call mpi_attr_put (comm, key, cachecomm, ierr)
else
  [ an error occurred ]
end if

return
end

```

Figure 15: Implementation of the BLACS routine `BLACS_GRIDINIT` showing how the BLACS communicators `commall`, `commrow`, and `commcol` are cached with the input communicator, `comm`.

```

subroutine igesd2d (comm, m, n, a, lda, rdest, cdest)
integer comm, m, n, lda, rdest, cdest
integer a(lda,*)
integer commall, ierr
integer akey, rkey, ckey
logical useflag
common /blacom/ akey, rkey, ckey
save /blacom/

call mpi_attr_get (comm, akey, commall, useflag, ierr)

if (useflag) then
  call mlacs_send (a, lda, m, n, MPI_INTEGER, rdest, cdest, 0, commall, ierr)
else
  [ an error occurred ]
end if

return
end

```

Figure 16: Implementation of the BLACS routine `IGESD2D`.

```

subroutine igebs2d (comm, scope, top, m, n, a, lda)
character*1 scope, top
integer m, n, lda
integer a(lda,*)
integer comm
integer cachecomm, ierr, key, nrow, ncol, myrow, mycol
logical useflag
integer akey, rkey, ckey
common /blacom/ akey, rkey, ckey
save /blacom/

call uncache_communicator (comm, scope, cachecomm, useflag, ierr)
if (useflag) then
  call mlacs_gridinfo (cachecomm, nrow, ncol, myrow, mycol, ierr)
  call mlacs_bcast (a, lda, m, n, MPI_INTEGER, myrow, mycol, cachecomm, error)
else
  [ an error occurred ]
end if

return
end

subroutine uncache_communicator (comm, scope, cachecomm, useflag, error)
integer comm, cachecomm, error
character*1 scope
logical useflag
integer key, ierr
integer akey, rkey, ckey
common /blacom/ akey, rkey, ckey
save /blacom/

if (scope .eq. 'A') then
  key = akey
else if (scope .eq. 'R') then
  key = rkey
else if (scope .eq. 'C') then
  key = ckey
end if

call mpi_get_attr (comm, key, cachecomm, useflag, ierr)

return
end

```

Figure 17: Implementation of the BLACS routine IGEBS2D.

```

subroutine igmin2d (comm, scope, top, m, n, a, lda, ra, ca, ldia, rdest, cdest)
character*1 scope, top
integer comm, m, n, lda, ldia, rdest, cdest
integer a(lda,*), ra(ldia,*), ca(ldia,*)
integer cachecom, coords(2), ierr
logical useflag
integer i, j, nrow, ncol, myrow, mycol
integer akey, rkey, ckey
common /blacom/ akey, rkey, ckey
save /blacom/
:
:
call uncache_communicator (comm, scope, cachecom, useflag, ierr)
if (useflag) then
  if (rdest .ne. -1) then
    call mlacs_reduce_minloc (a, a, lda, m, n, ra, ldia,
                             MPI_INTEGER, rdest, cdest, cachecom, iwork, work, ierr)
  else
    call mlacs_allreduce_minloc (a, a, lda, m, n, ra, ldia,
                                 MPI_INTEGER, cachecom, iwork, work, ierr)
  end if
do i=1,n
  do j=1,m
    call mpi_cart_coords (comm, ra(j,i), 2, coords)
    if (scope .eq. 'A') then
      ra(j,i) = coords(1)
      ca(j,i) = coords(2)
    else if (scope .eq. 'R') then
      ra(j,i) = myrow
      ca(j,i) = coords(1)
    else if (scope .eq. 'C') then
      ra(j,i) = coords(1)
      ca(j,i) = mycol
    end if
  end do
end do
else
  [ an error occurred ]
end if
return
end

```

Figure 18: Implementation of the BLACS routine IGMIN2D.