An MPI Implementation of the BLACS

V. R. Deshpande, W. B. Sawyer and D. W. Walker

LU for various grid sizes on thin-node SP-2

Solid line == MPI-BLACS (UTK)
Dashed line = MPI-BLACS (CSCS-ORNL)
Dotted line = MPL-BLACS

TR-96-11
May 1996
An MPI Implementation of the BLACS

V. R. Deshpande¹, W. B. Sawyer¹ and D. W. Walker²

TR-96-11, May 1996

Abstract. In this report, an MPI implementation of the Basic Linear Communication Subprograms (BLACS) is presented. A wide spectrum of MPI functionality has been used to implement BLACS as succinctly as possible, thus making the implementation concise, but still yielding good performance. We discuss some of the implementation details and present results for several different architectures with different MPI libraries. Finally, we gather our experiences in using MPI, and make some suggestions for the future functionality in MPI-2.

The MPI-BLACS library is available free under copyright for research purposes.

Keywords. MPI, BLACS, parallel architectures, libraries

This work was performed as part of the Joint CSCS/NEC Collaboration in Parallel Processing and will be presented at the MPI Developers Conference, 1996.

¹ Swiss Center for Scientific Computing (CSCS/SCSC-ETH), Via Cantonale, CH-6928 Manno, Switzerland
vaibhav@cscs.ch and sawyer@cscs.ch

² Dept. of Computer Science, University of Wales College of Cardiff, P O Box 916, Cardiff, Wales, CF2 3XF
David.W.Walker@cs.cf.ac.uk

CSCS/SCSC TECHNICAL REPORT
Contents

1 Introduction 3
2 Design 4
3 Implementation 5
  3.1 Grid Initialization 5
  3.2 Grid Coordinate Information 6
  3.3 Point-to-Point Communication 6
  3.4 Collective Communication 7
4 Results 8
5 MPI Evaluation 12

List of Figures

1 Performance of the ScaLAPACK LU factorization with our MPI-BLACS and Intel NX-BLACS on the Intel Paragon and also with our MPI-BLACS on the NEC Cenju-3. 10
2 Performance of the ScaLAPACK LU factorization on the IBM SP2 with our MPI-BLACS, UTK MPI-BLACS and MPL-BLACS 11
3 Implementation of BLACS_GRIDINIT 15
4 Implementation of BLACS_GRIDMAP 15
5 Macro for grid creation 16
6 Macros for Cartesian coordinate to system ID mappings 16
7 Macro to send a trapezoidal submatrix 17
8 Implementation of global summation routines 18
9 Implementation of global MAX routine 19
10 MLACS_REDUCE_LOC macro for combine routines 20

List of Tables

1 Performance of the ScaLAPACK LU factorization with our MPI-BLACS and Intel NX-BLACS on the Intel Paragon 8

1 Introduction

In this report an MPI ([MP95], implementation of the Basic Linear Algebra Communication Subprograms (BLACS) is presented. The BLACS are message passing routines that communicate matrices among processes arranged in a two-dimensional virtual process topology. It forms the basic communication layer for ScaLAPACK [CDPW94, CDO+94]. MPI provides the most suitable message-passing layer for BLACS, since it is widely available, has high level functionality to support the BLACS communication semantics as discussed in [DW95], and also has several advantages over other available communication libraries like PVM [GBD+94].

This implementation builds on the design suggested in [Wal94], which exploits high-level MPI routines to realize BLACS functionality succinctly. The BLACS communication context for the grid is defined as an MPI communicator which is created with a Cartesian topology. Its use ensures that messages meant for receipt in one phase of an application are not incorrectly received in another phase. In addition, communicators are defined for the BLACS communication context along a given row or column of the logical process grid, and are associated with the grid context as MPI attributes. General datatypes in MPI are used to describe the square and trapezoidal sub-matrices to be communicated. Extensive use is made of high-level MPI routines for collective communication, making the design clean and concise. We summarize the design of BLACS in Sect. 2.

In order to realize this basic design, numerous additional issues are addressed. The ability in the BLACS to assign an arbitrary system process to a position in the grid is elegantly realized, requiring a permutation of processes from the default row-major ordering in MPI. In addition, recent alterations to the BLACS interfaces [DW95] necessitated the change of some parts of the original design. Efficiency of the implementation is also addressed, resulting in the use of macros which inherit the functionality of the previously defined MPI Linear Algebra Communication Subroutines (MLACS). The implementation is discussed in Sect. 3.

At least one other MPI implementation [Wha95b] of the BLACS is currently available. This implementation is highly optimized and makes extensive use of the topology information, which can either be provided upon compilation of the library, or when the BLACS collective communication routines are called. Such an approach is necessary for simpler message-passing paradigms such as Intel NX and exploits basic sends and receives to implement collective communication. On the other hand, one should anticipate that in future, all levels of an MPI implementation are optimized for a given architecture, indicating the close mapping of the BLACS routines to the corresponding high-level MPI routines. The importance of ScaLAPACK warrants a look at different approaches to the MPI implementation of BLACS and thus the issues involved in both versions are discussed in Sect. 4 and some conclusions are drawn from available performance data.

In addition to its use in supporting ScaLAPACK, our BLACS library has proven to be a demanding test of various MPI implementations on different architectures, due to the wide spectrum of functionality used. In Sect. 5 we discuss the problems encountered and indicate possible new functionality for MPI-2, such as better support for Cartesian topologies and dynamic process allocation.

AN MPI IMPLEMENTATION OF THE BLACS

TR-96-11, MAY 1996
2 Design

In [Wal94] Walker suggests an implementation of the BLACS as a layer built on the MPI Linear Algebra Communication Subprograms (MLACS). MLACS make use of the extensive functionality available in MPI for process groups, communicator management, and attribute caching. In this design, the communication context for a given virtual process mesh corresponds to an MPI communicator \( A \) having two-dimensional Cartesian topology. As communication may take place along any given row or column of the process grid, communicators \( R \) and \( C \) with a one-dimensional Cartesian topology are defined which contain all processes in given row or column, respectively. Use is made of MPI's caching facility to associate these communicators with the communicator \( A \).

BLACS allows a flexible mapping of system process numbers to positions in the virtual process grid, thus allowing the user to exploit the underlying physical processor configuration. Not only does BLACS_GRIDINIT allow row-major or column-major ordering of the system processes, but BLACS_GRIDMAP allows an arbitrary mapping of the system processes to grid positions. On the other hand, MPI's Cartesian coordinate functionality is based on row-major ordering. The column-major and arbitrary mappings can be realized with the MPLCOMM_SPLIT routine, which creates a new communicator with permuted ranks. The Cartesian topology is then imposed on this new communicator as indicated previously.

The BLACS point-to-point routines can be implemented by using \( A \) with appropriate process ranks for the sender and receiver, which can be determined by Cartesian coordinate conversion routines in MPI. MPI datatypes are defined for the square or trapezoidal matrices which are to be communicated. One element of a given type is then sent or received.

BLACS broadcast routines require a scope, namely either all processes in the grid, or all those in a given row or column. Depending on the scope, either MPLBCAST is used with the communicator \( A \) and the appropriate root process, or the \( R \) or \( C \) communicator is first uncached, and then a corresponding broadcast with that communicator is performed.

The BLACS combine routines are implemented in a way similar to the BLACS broadcast routines except that use is made of MPI global reduce operations. A combination of predefined and user-defined operators fulfills the task of performing the required operation and putting the result on the required process.

3 Implementation

The implementation of MPI-MLACS is based on MLACS as suggested in [Wal94]. In order to gain efficiency however, the MLACS routines were implemented as macros, and thus the corresponding MLACS code is "inlined" in the optimized version. This offers an improvement in performance without the loss of the clean structure which the MLACS prototype provides.

The fact that some MPI implementations do not allow the same pointer to be used for the input and output buffers in collective communication routines\(^1\) (e.g., MPLREDUCE) necessitates the use of dynamic memory allocation for the combine routines MPLSUM2D, MPLMAX2D, and MPLMIN2D. In addition, dynamically allocated work arrays are needed for the point-to-point and broadcast routines dealing with trapezoidal matrices. Considering these requirements, we assume the availability of a Fortran 90 compiler, or at least a Fortran 77 compiler which supports dynamic memory allocation.

The resulting BLACS implementation is highly compact, consisting of less than a few thousand lines of code. This succinctness is due, on the one hand, to the use of high level MPI functionality to support Cartesian grids, process groups, attributes and user-defined types and operators. On the other hand, the macro language GNU m4 [Sei94] is used to allow generation of the Fortran routines for all data types, and to insert debugging macros, etc. Each communication routine consists of a single macro, and thus it is easy to make any changes to the library.

The error handling mechanism implemented in BLACS makes use of the MPLABORT routine to abort the particular process with a specific error code.

Although our BLACS implementation follows the MLACS design closely, certain routines were restructured considerably, and others which were not included in [Wal94] where designed and implemented. We subsequently discuss these additions individually in the following sections.

3.1 Grid Initialization

The BLACS have two general purpose routines BLACS_SET and BLACS_GET to set and obtain information about various BLACS internals. These routines are commonly used to retrieve a default system context for input into BLACS_GRIDINIT or BLACS_GRIDMAP. For it to include all the existing MPI processes, the default system context returned in BLACS_GET corresponded to MPLCOMM_WORLD.

For grid initialization itself, an additional complexity was incurred with the most recent release of the BLACS User's Guide [DW95], which specified that BLACS_GRIDINIT should support column-major mapping along with the previously specified row-major mapping of system processes to grid locations. Since MPI's Cartesian topology supports only row-major ordering, BLACS_GRIDINIT's implementation therefore requires the permutation of system process ranks, just as BLACS_GRIDMAP did.

The similarity of BLACS_GRIDINIT (Fig. 3) and BLACS_GRIDMAP (Fig. 4) is apparent. Both rely on the macro MLACS_GRIDCREATE (Fig. 5) to perform the permutation, create the communicators \( A, R \) and \( C \) with the proper Cartesian topology, and cache \( R \) and \( C \).

\(^1\)This requirement is not clearly stated in the MPI standard, but is imposed in MPICH, et al.
The extra processes which do not constitute a grid are returned with `MPI_COMM_NULL` and are carried until `BLACS-GRIDEXIT` is called. These processes cannot participate in any MPI calls for the given grid, therefore it is mandatory to check for `MPI_COMM_NULL` in every BLACS communication routine.

Despite the underlying complexity of these operations, the resulting code is concise due to the availability of Cartesian topologies, communicator “splitting” and attribute caching in MPI.

### 3.2 Grid Coordinate Information

The permutation of system process numbers introduces an additional problem, namely that `BLACS_PCOORD` must return the proper coordinates of a given system process and `BLACS_PNUM` the system process number of the given coordinates. Clearly these operations must take the mapping introduced by `BLACS_GRIDINIT` or `BLACS_GRIDMAP` into account.

The mapping information can be retrieved with `MPI_GROUP_TRANSLATE_RANKS` as illustrated in Fig. 6. The MPI group `groupworld` is defined in the initialization of BLACS as the group of processes in `MPI_COMM_WORLD`. The actual MPI Cartesian mapping routines `MPI_CART_COORDS` and `MPI_CART_RANK` assume a row-major ordering of the process ranks within `group` and, although they do not incur a large overhead, could be replaced by the corresponding mapping.

### 3.3 Point-to-Point Communication

A point-to-point communication consists of the following stages,

1. Check communicator, validate input parameters;
2. Translate between process coordinates and rank;
3. Create data type for the communication;
4. Call MPI routine to communicate data;
5. Free the general datatype.

Point-to-point communication makes use of the communication context `A` to transfer a trapezoidal or rectangular matrix between two processes in the corresponding grid. Since `A` is itself an MPI communicator, there is no need to uncache it, thus slightly reducing the overhead inherent in [Wal94]. Translating from process coordinates to rank is performed in `INITIALIZE_POINT` which ultimately calls the routine `MPI_CART_RANK`.

The actual send primitive is the buffered-mode `MPI_BSEND` operation, which avoids potential deadlock situations that may arise from cyclic communication patterns. These local completion semantics correspond to the locally blocking mechanism required by [DW98]. `MPI_BSEND` does however need buffer space allocated with `MPI_BBUF_ATTACH` during the initialization of BLACS which is then deallocated in `BLACS_EXIT`.

This mechanism is illustrated in the `MLACS_SEND_TRAP` macro used by `ZTRBS2D` in Fig. 7. In this case, a stencil for the MPI datatype for a trapezoidal matrix is created in `SETUP_INDEXED`, which is then manipulated with the routines `MPI_TYPE_INDEXED`, `MPI_TYPE_COMMIT`, and freed later with `MPI_TYPE_FREE`.

### 3.4 Collective Communication

While BLACS’ broadcast routines `ZGEB2D`, `ZGEBR2D`, `ZTRBS2D` and `ZTRBR2D` are implemented as suggested in [Wal94], the reduction routines `ZGAMX2D`, `ZGAMN2D` and `ZGSUM2D` required redesign.

A temporary, dynamically allocated work array was introduced in the global summation `ZGSUM2D` (Fig. 8), since in some MPI libraries, the underlying `MPI_ALLREDUCE` and `MPI_REDUCE` require that the pointers to the input and output buffers be different. The macros `MLACS_REDUCE` and `MLACS_ALLREDUCE` perform the combine operation on the root `(rroot, croot)` or on every processor, respectively.

The `ZGAMX2D` and `ZGAMN2D` were redefined slightly in [DW95] from the original `ZGMAX2D` and `ZGMIN2D`. The use of the `MPI_LOCAL_LOC` operation provided for `MPI_REDUCE` as suggested in [Wal94] is not sufficient. Instead a compare operation for the absolute value of all types is now defined using the `MPI_OP_CREATE` primitive. The implementation of the absolute maximum BLACS routines can be seen in Fig. 9. The `MLACS_REDUCE_LOC` macro, which determines the location of the maximum or minimum along with the supremum itself, is illustrated in Fig. 10.
4 Results

The performance of the BLACS library was investigated on the IBM SP2 with the MPICH [GLDS95], Intel Paragon (with the recently developed NMPI library) and the NEC Cenju-3 with the prototype MPI/DE [KTK95] library using the ScaLAPACK LU factorization routine.

Figure 1 shows the performance of the LU factorization on the Intel Paragon which is an 860 XPS/22MP model with three 75 MFlop/s Processors per node with 64MB memory. The CPU includes 16 KB data and instruction caches, and the bandwidth between the floating point unit and the data cache memory peaks at 1.2 GB/sec. The latency is 41 microseconds with a bandwidth of 130 MB/sec. The standard OSF UNIX kernel is used for all tests.

Our MPI-BLACS library was compared against the native Intel NX-BLACS based on the underlying NX communication layer for optimum block size (ranging from 8 to 20). The Paragon performance is significantly less than that reported in [CDO+94], the reasons for which are not known.

For the optimal mesh, performance of the MPI-BLACS and NX-BLACS version deviate only slightly, indicating that the overhead of NMPI, which is based on NX is small. Non-optimal mesh sizes reveal a much larger discrepancy in performance as shown in Table 1, possibly indicating the lack of a pipelining topology [Wha95a] in our version.

Table 1: Performance of the ScaLAPACK LU factorization with our MPI-BLACS library and Intel NX-BLACS on the Intel Paragon.

<table>
<thead>
<tr>
<th>Grid PxQ</th>
<th>NX-BLACS N MFlop/s</th>
<th>MPI-BLACS N MFlop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>500</td>
<td>76.69</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>290.03</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>380.40</td>
</tr>
<tr>
<td>2x16</td>
<td>500</td>
<td>97.54</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>283.79</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>619.12</td>
</tr>
<tr>
<td>4x8</td>
<td>500</td>
<td>85.74</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>263.63</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>568.83</td>
</tr>
</tbody>
</table>

Figure 1 also shows the performance with optimal block size with our MPI BLACS library on the NEC Cenju-3 machine. Cenju-3 is a distributed memory parallel machine with up to 256 processor elements, each having a MIPS R4400 running at 75MHz clock and local memory connected by multistage network based on 4x4 switches. The point-to-point throughput of the network is 40 Mbytes/sec. The MPI implementation on Cenju-3, called MPI/DE [KTK95], is a part of a parallel operating system DenEn based on CMU Mach microkernel. It attains 40 microseconds for the minimum latency and 20 Mbytes/sec for the maximum throughput. These performances are modest but should be considered in light of the prototype nature of the DenEn operating system and the MPI/DE library. Also, in the absence of a Fortran compiler which could support dynamic memory allocation in some of the BLACS routines, the library was passed through a Fortran to C converter and compiled using a C compiler which might have affected the performance.

Finally, our MPI-BLACS implementation has been compared with the BLACS version described in [Wha95b] using both, MPICH and native MPL for message-passing on the IBM SP2. The results in Fig. 2 indicate that the additional complexity, e.g., the support of pipelining topologies, of the latter indeed provides some additional performance. The absence of such a support for topology in our BLACS implementation is based on the assumption that the MPICH library is optimized for all high-level collective communication routines on the IBM SP2 which may not be appropriate.

The native MPL-BLACS results are considerably better than either MPI version, except for very large problem sizes, where a limitation on message buffer size might play a role.

We feel the positive performance results indicate that it is possible to implement a complicated software package like BLACS succinctly, efficiently, and allowing for portability using the wide spectrum of MPI functionality.
Figure 1: Performance of the ScaLAPACK LU factorization with our MPI-BLACS and Intel NX-BLACS on the Intel Paragon and also with our MPI-BLACS on the NEC Cenju-3.

Figure 2: Performance of the ScaLAPACK LU factorization on the IBM SP2 with three different versions of BLACS: MPI-BLACS developed by R. Clint Whaley (UTK), MPI-BLACS designed and implemented by the authors, and MPI-BLACS using native message-passing. Both MPI-BLACS versions have similar performance, slightly less than the MPI-BLACS version, except for large problem sizes, for which MPI appears to have performance degradation due to sending large messages.
5 MPI Evaluation

Our experiences with MPI were positive — no other message-passing library could have provided the functionality to implement the BLACS in so few lines of code. Indeed, certain functionality is crucial for proper implementation. For example, in MPI-BLACS, the matrix parameters $M$ and $N$ can be varied by the user as long as $M \times N$ is same on all processors, unlike in PVM where the data must be unpacked in the same manner as it is packed (see discussion in [DW95]), restricting the means of changing the shape of the matrix by varying only its leading dimension.

During the design, implementation and benchmarking of BLACS on the above mentioned machines using different MPI libraries, we encountered some MPI design and implementation issues which, if improved upon, could allow a still cleaner implementation of the BLACS library, and also might improve the performance.

- As mentioned earlier, MPI's Cartesian coordinate functionality supports only row-major ordering. This makes the implementation of column-major and arbitrary mappings more complex affecting the performance due to the need of permutation of ranks. Although this can be handled easily by MPI_GROUP_TRANSLATE_RANKS, it could be more generally implemented by allowing a user mapping in a new function, e.g., MPI_CART_MAP, or by extending the reorder capability in the existing MPI_CART_CREATE.

- In ScaLAPACK, there is a need to create grids with fewer processes from the number of available processes. Those processes which do not form a grid are carried throughout since there is no provision in MPI to handle such processes effectively. The incorporation of dynamic allocation or spawning of processes as suggested in ([MP96]) should offer a much cleaner and more efficient alternative to the present design.

- According to ([MP96]) it is not possible to input MPI_COMM_NULL, or any other null handle arguments to MPI routines, even though this is allowed in some MPI libraries. There is no clear rational given for this and we feel it is an unnecessary limitation which should be removed in MPI-2.

- In our BLACS implementation, there is no explicit support for the TOP (topology) argument which emulates the underlying network topology during communication. The assumption that MPI is optimal for a given architecture justifies that there is no need for one to know beforehand which topology is optimal for a given communication.

To achieve maximum performance, we expect that in future, the MPI library on every architecture will be optimized for all communication. Indeed, by default the MPI-BLACS version ([Wha95b]) developed at UTK, by default makes use of the most direct MPI functionality to implement BLACS communication, i.e., MPI_BCAST for e.g., etc.

If feasible, we propose to augment the MPI functionality to exploit user knowledge of the underlying topology and reconfiguring certain high-level communication routines at run-time to improve performance. If such functionality is realistically implementable, we would recommend its integration into MPI-2.

In addition to the above suggestions about potential improvements to the MPI standard, our work yielded some experiences about existing MPI implementations, which are summed below:

- In some MPI implementations, the input and output buffers in global reduction operations like MPI_ALLREDUCE cannot be same, requiring the allocation of extra memory. While this limitation is perhaps necessary for more complex collective communication routines, e.g., MPI_ALLTOALLV, it is not specifically stated in ([MP95]). In fact several MPI implementations do not have such a restriction. We feel that the use of the same buffer should be allowed in routines where it can be cleanly implemented, and any limitations in its use need to be documented in the MPI standard.

- In some MPI implementations, functionality for the derived data types or for communication cannot handle arguments of length zero. This necessitates checking the arguments for their length before passing them to the appropriate functions. This limitation is not documented in ([MP95]) and, in our opinion should not be imposed.

In addition to the above experiences, ([Wha95c]) discusses several proposed extensions to MPI like interface inter-operability, communicator formation and non-blocking communication which are also of particular interest to us, and should be considered for integration into MPI-2.

Acknowledgments We would like to thank R. C. Whaley for his valuable comments to the design and implementation of MPI-BLACS and his benchmarking efforts on the IBM SP2. In addition, we would like to thank T. Nakata for his benchmarking on the NEC Cenju-3 running the prototype Mach kernel. Finally we gratefully acknowledge the financial assistance of the NEC Corporation during the realization of this project.
References


Figure 3: Implementation of BLACS_GRIDINIT

```fortran
subroutine BLACS_GRIDINIT(icontxt, order, nprow, npcol)
implicit none
integer  icontxt, nprow, npcol
character*1 order
integer mynprow, mynpcol, mynpnum, nprocs, newrank, color, ierr
include "mpif.h"
include "mlacs.h"
MLACS_INIT
MLACS_PROCINFO
if (nprocs .ge. nprow*npcol) then
  color = MPI_UNDEFINED
  if (mynum .lt. nprow*npcol) then
    if (order .eq. "C" .or. order .eq. "c") then
      newrank = (mynum/nprow) + mod(mynum, nprow)*npcol
    elseif (order .eq. "R" .or. order .eq. "r") then
      newrank = mynum
    else
      newrank = mynum
    end if
    color = 0
  end if
  MLACS_GRID_CREATE
else
  ierr = -1
endif
MLACS_DEBUG(ierr, 'ne.', 0, BLACS_GRIDINIT_ERR)
return
```

Figure 4: Implementation of BLACS_GRIDMAP

```fortran
subroutine BLACS_GRIDMAP(icontxt, usermap, ldu, nprow, npcol)
implicit none
integer  icontxt, ldu, nprow, npcol
integer usermap(ldu,*)
integer myprow, mypcol, mypnum, nprocs, newrank, color, ierr
integer ii, jj
include "mpif.h"
include "mlacs.h"
MLACS_INIT
MLACS_PROCINFO
if (nprocs .ge. nprow*npcol) then
  color = MPI_UNDEFINED
  do jj = 1, npcol
    do ii = 1, nprow
      if ( (mynum .eq. usermap(ii, jj)) ) then
        newrank = ii + (jj-1) + npcol + (j-1) .ne. 0
      else
        newrank = 0
      end if
    end do
  end do
endif
MLACS_DEBUG(ierr, 'ne.', 0, BLACS_GRIDMAP_ERR)
return
```

CSCS/SCSC Technical Report

TR-96-11, MAY 1996
Figure 5: Macro for grid creation

```fortran
define(MLACS_GRID_CREATE,
   dims(1) = nprow
   dims(2) = npcol
   reorder = .FALSE.
   periods(1) = .FALSE.
   periods(2) = .FALSE.
   if (icontxt .ne. MPI_COMM_NULL) then
      MPI_CALL('MPI_COMM_SPLIT (icontxt, color, newrank, comm, ierror)')
      if (comm .ne. MPI_COMM_NULL) then
         MPI_CALL('MPI_CART_CREATE(comm, 2, dims, periods, reorder, &
            icontxt, ierror)')
      end if
      MPI_CALL('MPI_COMM_FREE (comm, ierror)')
   end if
)
```

Figure 6: Macros for Cartesian coordinate to system ID mappings

```fortran
define(MLACS_CART_COORDS,
   if (icontxt .eq. MPI_COMM_NULL) then
      prow = -1
      pcol = -1
   else
      MLACS_CART_2D_TEST(icontxt, ierror)
      MPI_CALL('MPI_COMM_GROUP (icontxt, group, ierror)')
      coords(1) = prow
      coords(2) = pcol
   end if
)
```

```fortran
define(MLACS_CART_RANK,
   if (icontxt .eq. MPI_COMM_NULL) then
      pnum = -1
   else
      MLACS_CART_2D_TEST(icontxt, ierror)
      MPI_CALL('MPI_CART_RANK(icontxt, coords, rankall, ierror)')
      coords(1) = prow
      coords(2) = pcol
   end if
)
```

Figure 7: Macro to send a trapezoidal submatrix

```fortran
define(MLACS_SEND_TRAP,
   if (icontxt .eq. MPI_COMM_NULL) return
   INITIALIZE_POINT(icontxt, rdest, cdest, rank, ierror)
   SETUP_INDEXED
   nn = n
   if (m.lt.n .and. (diag.eq."U".or. diag.eq."U") then
      nn = n - 1
   endif
   MPI_CALL('MPI_TYPE_INDEXED(nn, workl, work2, datatype, &
      mtype, ierror)')
   HPI_CALL('MPI_TYPE_COMMIT(mtype, ierror)')
   MPI_CALL('MPI_BSEND (a, 1, mtype, rank, tag, icontxt, ierror)')
   MPI_CALL('MPI_TYPE_FREE (mtype, ierror)')
)
```
Figure 8: Implementation of global summation routines

```fortran
subroutine $1GSUM2D (icontxt, scope, top, m, n, lds, & rroot, croot)
  implicit none
  integer icontxt, m, n, lds, rroot, croot
  real (kind=8) s(lds,:), work(n)
  integer nprow, npcol, datatype, optype
  integer myprow, mypcol
  include "mpif.h"
  include "mlacs.h"
  if(m.eq.0.or.n.eq.0) return

  datatype = $3
  optype = MPI_SUM
  if (rroot .ge. 0 .and. croot .ge. 0) then
    MLACSJIEDUCE
  else if (rroot .eq. -1 .or. croot .eq. -1) then
    MLACS_ALLREDUCE
  else
    ierror = -1
  end if
  MLACS_DEBUG(ierr, '.ne.', 0, $1GSUM2D_ERR)
  return
end
```

Figure 9: Implementation of global MAX routine

```fortran
subroutine $1GAMX2D (icontxt, scope, top, m, n, lds, ra, ca, & rcflag, rroot, croot)
  implicit none
  integer icontxt, m, n, lds, raflag, rroot, croot
  real (kind=8) a(lds,:), work(n)
  integer ra(raflag,:), ca(raflag,:), iwrk(n)
  integer nprow, npcol, datatype, optype
  integer myprow, mypcol
  include "mpif.h"
  include "mlacs.h"
  if(m.eq.0.or.n.eq.0) return

  datatype = $3
  optype = $4
  if (raflag .gt. 0) then
    MLACS_GRIDINFO
    if (rroot .ge. 0 .and. croot .ge. 0) then
      MLACS_REDUCE_LOC
    else if (rroot .eq. -1 .or. croot .eq. -1) then
      MLACS_ALLREDUCE_LOC
    else
      ierror = -1
    end if
    else if (raflag .eq. -1) then
      if (rroot .ge. 0,.and. croot .ge. 0) then
        MLACS_REDUCE
      else if (rroot .eq. -1 .or. croot .eq. -1) then
        MLACS_ALLREDUCE
      else
        ierror = -1
      end if
    end if
    else
      ierror = -1
    end if
  else
    ierror = -1
  end if
  MLACS_DEBUG(ierr, '.ne.', 0, $1GAMX2D_ERR)
  return
end
```
Figure 10: MLACS_REDUCE_LOC macro for combine routines

```fortran
if (icontext .eq. MPI_COMM_NULL) return
INITIALIZE_COLLECTIVE(icontext, scope, rroot, croot, rank, comm, ierr)
MPI_CALL('MPI_COMM_RANK(comm, locid, ierr)')
do i = 1, n
  MPI_CALL('MPI_ALLREDUCE(a(l,i), work, m, datatype, optype, &
             comm, ierr)')
do j = 1, m
    ra(j,i) = -1
    ca(j,i) = -1
    if (a(j,i) .eq. work(j)) then
      ra(j,i) = myprow
      ca(j,i) = mypcol
    end if
    if (locid .eq. rank) then
      a(j,i) = work(j)
    end if
  end do
  MPI_CALL('MPI_REDUCE(ra(l,i), iwrk, m, MPI_INTEGER, MPI_MAX, &
                  rank, comm, ierr)')
do j = 1, m
    if (locid .eq. rank) then
      ra(j,i) = iwrk(j)
    end if
  end do
  MPI_CALL('MPI_REDUCE(ca(l,i), iwrk, m, MPI_INTEGER, MPI_MAX, &
                  rank, comm, ierr)')
do j = 1, m
    if (locid .eq. rank) then
      ca(j,i) = iwrk(j)
    end if
  end do
end do
```

1995


TR-95-06 P. Ackermann and U. Meyer: Prototypes for Audio and Video Processing in a Scientific Visualization Environment based on the MET++ Multimedia Application Framework. (June 1995)


1996


TR-96-03 Brian J. N. Wylie and Akiohi Endo: Annai/PMA Multi-level Hierarchical Parallel Program Performance Engineering. (April 1996)

TR-96-04 Christian Cléménçon, Akiohi Endo, Josef Fritscher, Andreas Müller, and Brian J. N. Wylie: Annai Scalable Run-time Support for Interactive Debugging and Performance Analysis of Large-scale Parallel Programs. (April 1996)


TR-96-06 W. P. Petersen: Some evaluations of Random Number Generators in real*8 Format. (April 1996)

TR-96-07 Tetsuya Takashii: Heavy quark potential and effective actions on blocked configurations. (April 1996)

TR-96-08 Norio Masuda and Frank Zimmermann: PRNGlib: A Parallel Random Number Generator Library. (May 1996)


TR-96-10 Edgar A. Gertiesen: Automatized Generation of Block-Structured Meshes for a Parametric Geometry. (May 1996)