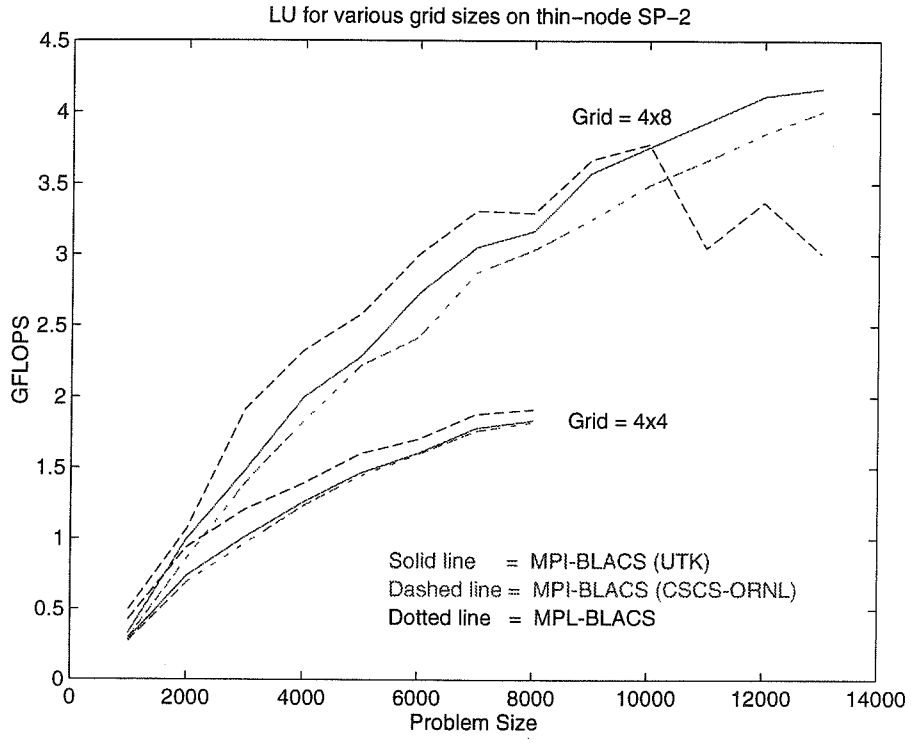
Centro Svizzero di
Calcolo Scientifico

Swiss Center for
Scientific Computing

ETH

An MPI Implementation of the BLACS

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



TECHNICAL REPORT

TR-96-11

May 1996

OTHER PUBLICATIONS BY CSCS/SCSC

Annual Report:	yearly review of activities and projects
CrosSCutS (triannually):	newsletter featuring announcements relevant to our users as well as research highlights in the field of high-performance computing
Speedup Journal (biannually):	proceedings of the SPEEDUP Workshops on Vector and Parallel Computing, published on behalf of the SPEEDUP Society
User's Guide:	manual to hardware and software at CSCS/SCSC

To receive one or more of these publications, please send your full name and complete address to:

Library
CSCS/SCSC
via Cantonale
CH-6928 Manno
Switzerland

Fax: +41 (91) 610 8282

E-mail: `library@cscs.ch`

Technical Reports are also available from:
`http://www.cscs.ch/Official/Publications.html`
A list of former IPS Research Reports is available from:
`http://www.cscs.ch/Official/IPSreports.html`

An MPI Implementation of the BLACS

V. R. Deshpande[1], W. B. Sawyer[1] and D. W. Walker[2]

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`

Contents

List of Figures

List of Tables

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 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.

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 $\mathcal{A}$ having two-dimensional Cartesian topology. As communication may take place along any given row or column of the process grid, communicators $\mathcal{R}$ and $\mathcal{C}$ 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 $\mathcal{A}$.

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 MPI_COMM_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 $\mathcal{A}$ 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 MPI_BCAST is used with the communicator $\mathcal{A}$ and the appropriate root process, or the $\mathcal{R}$ or $\mathcal{C}$ 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-BLACS 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., MPI_REDUCE) necessitates the use of dynamic memory allocation for the combine routines *x*GSUM2D, *x*GAMX2D, and *x*GAMN2D. 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 MPI_ABORT 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 MPI_COMM_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 $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{C}$ with the proper Cartesian topology, and cache $\mathcal{R}$ and $\mathcal{C}$.

[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 a 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,

(i) check communicator, validate input parameters; (ii) translate between process coordinates and rank; (iii) create data type for the communication; (iv) call MPI routine to communicate data; (v) free the general datatype.

Point-to-point communication makes use of the communication context $\mathcal{A}$ to transfer a trapezoidal or rectangular matrix between two processes in the corresponding grid. Since $\mathcal{A}$ 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 [DW95]. MPI_BSEND does however need buffer space allocated with MPI_BUFFER_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 *x*TRSD2D 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 *x*GEBS2D, *x*GEBR2D, *x*TRBS2D and *x*TRBR2D are implemented as suggested in [Wal94], the reduction routines *x*GAMX2D, *x*GAMN2D and *x*GSUM2D required redesign.

A temporary, dynamically allocated work array was introduced in the global summation *x*GSUM2D (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 *x*GAMX2D and *x*GAMN2D were redefined slightly in [DW95] from the original *x*GMAX2D and *x*GMIN2D. The use of the `MINLOC` 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 i860 XP/S-22MP model with three 75 MFlop/s Processors per node with 64MB memory. The CPU includes separate 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 microsec 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.

Grid PxQ	N	**NX-BLACS** MFlop/s	**MPI-BLACS** MFlop/s
	500	76.09	40.13
4x4	1000	204.03	123.06
	2000	380.40	266.80
	500	97.54	53.52
2x16	1000	283.79	162.04
	2000	619.12	394.38
	500	85.74	42.63
4x8	1000	263.63	142.67
	2000	569.83	350.76

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.

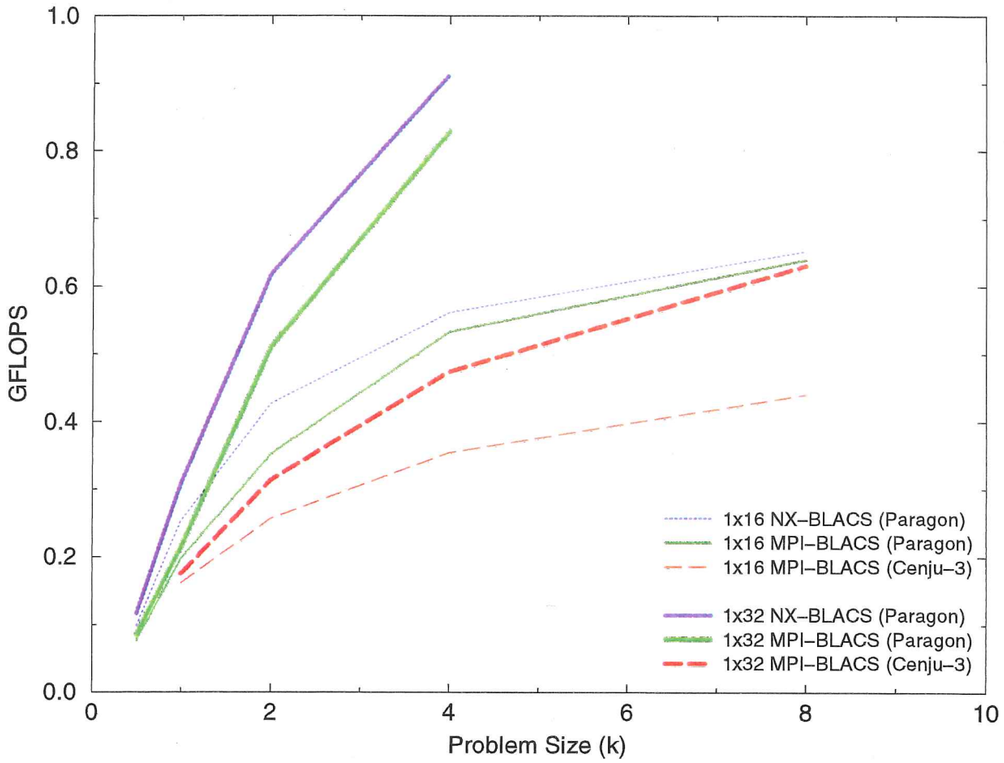


Figure 1: Performance of the ScaLAPACK *LU* factorization with our MPI–BLACS and Intel NX–BLACS on the Intel Paragon and also with our MPI–BLACS on the NEC Cenju-3.

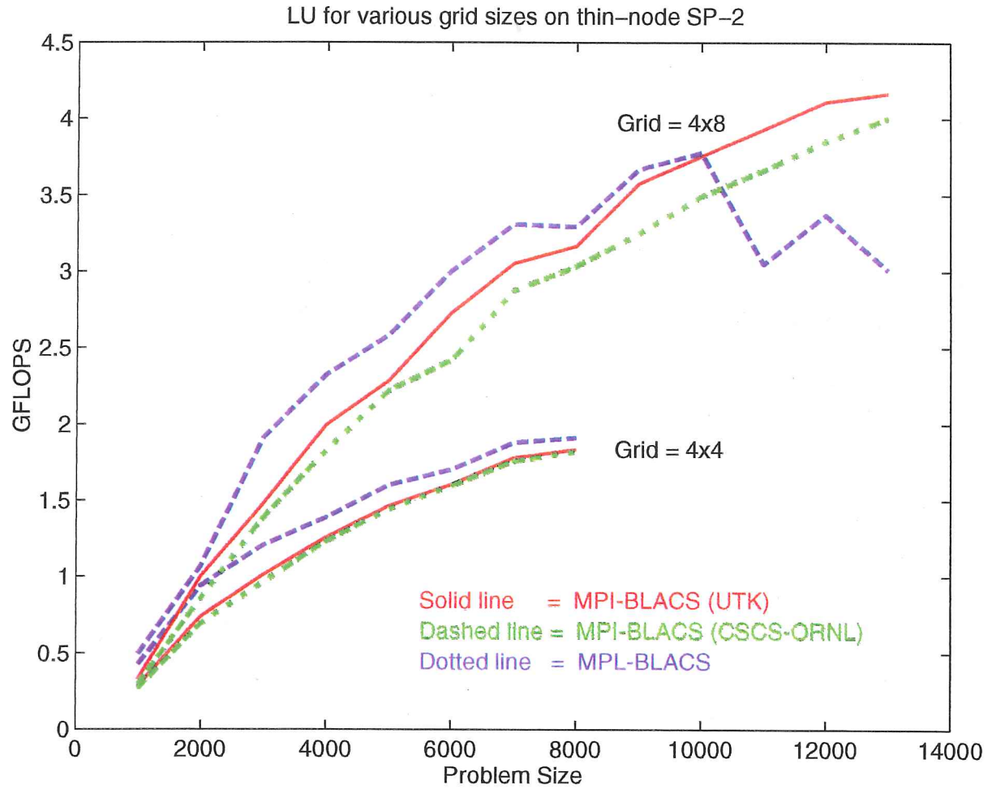


Figure 2: Performance of the ScaLAPACK *LU* factorization on the IBM SP2 with three different versions of BLACS: MPI-BLACS developed by R. Clint Whaley (UTK), MPI-BLACS designed and implemented by the authors, and MPL-BLACS using native message-passing. Both MPI-BLACS versions have similar performance, slightly less than the MPL-BLACS version, except for large problem sizes, for which MPL 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 [(MP95] 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 *x*`GEBS2D`, 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

[CDO+94] Jacyoung Choi, Jack J. Dongarra, L. Susan Ostrouchov, Antoine P. Petitet, David W. Walker, and R. Clint Whaley. The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines. Technical Report ORNL/TM-12470, Oak Ridge National Laboratory, September 1994.

[CDPW94] Jaeyoung Choi, Jack J. Dongarra, Roldan Pozo, and David W. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. LAPACK Working Note 55, 1994.

[DW95] Jack J. Dongarra and R. Clint Whaley. A user's guide to BLACS v1.0. LAPACK Working Note DRAFT, ORNL, 1995.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[GLDS95] Bill Gropp, Rusty Lusk, Nathan Doss, and Tony Skjellum. MPICH — a high-performance portable implementation of MPI. June 1995. Presented at the MPI Developers Conference.

[KTK95] Koichi Konishi, Yosuke Takano, and Akihiko Konagaya. MPI/DE: an MPI library for Cenju-3. June 1995. Presented at the MPI Developers Conference.

[(MP95] (MPIF) Message Passing Interface Forum. MPI: a Message-Passing Interface standard (version 1.1). Revision of article appearing in the *International Journal of Supercomputing Applications*, 8(3/4):157–416, 1994, June 1995.

[(MP96] (MPIF) Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface (DRAFT). 1996.

[Sei94] René Seindal. *GNU m4, A simple macro-processor.* 1994.

[Wal94] David W. Walker. An MPI version of the BLACS. 1994. Presented at the Scalable Parallel Libraries Conference.

[Wha95a] R. Clint Whaley. Private correspondence, 1995.

[Wha95b] R. Clint Whaley. Installing and testing the BLACS. LAPACK Working Note DRAFT, University of Tennessee, 1995.

[Wha95c] R. Clint Whaley. Some plebian extensions to MPI. Working Note DRAFT, University of Tennessee, 1995.

Figure 3: Implementation of BLACS_GRIDINIT

```
      subroutine BLACS_GRIDINIT(icontxt, order, nprow, npcol)
      implicit    none
      integer     icontxt, nprow, npcol
      character*1 order
      integer  myprow, mypcol, mypnum, nprocs, newrank, color, ierror

      include "mpif.h"
      include "mlacs.h"

      MLACS_INIT
      MLACS_PROCINFO

      if (nprocs .ge. nprow*npcol) then
        color = MPI_UNDEFINED
        if (mypnum .lt. nprow*npcol) then
          if (order .eq. "C" .or. order .eq. "c") then
            newrank = (mypnum/nprow) + mod(mypnum, nprow)*npcol
          elseif (order.eq. "R" .or. order .eq. "r") then
            newrank = mypnum
          else
            newrank = mypnum
          end if
          color = 0
        end if
        MLACS_GRID_CREATE
      else
        ierror = -1
      endif
      MLACS_DEBUG(ierror, `.ne.', 0, BLACS_GRIDINIT_ERR)
      return
      end
```

Figure 4: Implementation of BLACS_GRIDMAP

```
      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, ierror
      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 ( mypnum .eq. usermap( ii, jj ) ) then
              newrank = ( ii - 1 ) * npcol + ( jj - 1 )
              color = 0
            end if
          end do
        end do
        MLACS_GRID_CREATE
      else
        ierror = -1
      endif
      MLACS_DEBUG(ierror, `.ne.', 0, BLACS_GRIDMAP_ERR)
      return
      end
```

Figure 5: Macro for grid creation

```
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)´)
          MPI_CALL(`MPI_COMM_FREE (comm, ierror)´)
          if (color .eq. 0) then
            MLACS_TOPO_TEST(icontxt, status, ierror)
            MLACS_CART_2D_TEST(icontxt, ierror)

            remaindims(1) = .FALSE.
            remaindims(2) = .TRUE.
            MPI_CALL(`MPI_CART_SUB (icontxt, remaindims, comm, ierror)´)
            CACHE_COMMUNICATOR(icontxt,rkey,comm,ierror)

            remaindims(1) = .TRUE.
            remaindims(2) = .FALSE.
            MPI_CALL(`MPI_CART_SUB(icontxt, remaindims, comm, ierror)´)
            CACHE_COMMUNICATOR(icontxt,ckey,comm,ierror)
          end if
        else
          icontxt = MPI_COMM_NULL
        end if
      end if
´)
```

Figure 6: Macros for Cartesian coordinate to system ID mappings

```
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)´)
        MPI_CALL(`MPI_GROUP_TRANSLATE_RANKS (groupworld, 1, pnum,
     &                               group, rankall, ierror)´)
        MPI_CALL(`MPI_CART_COORDS(icontxt, rankall, 2, coords, ierror)´)
        prow = coords(1)
        pcol = coords(2)
      end if
´)

define(MLACS_CART_RANK,`
      if (icontxt .eq. MPI_COMM_NULL) then
        pnum = -1
      else
        MLACS_CART_2D_TEST(icontxt, ierror)
        coords(1)=prow
        coords(2)=pcol
        MPI_CALL(`MPI_CART_RANK(icontxt, coords, rankall, ierror)´)
        MPI_CALL(`MPI_COMM_GROUP (icontxt, group, ierror)´)
        MPI_CALL(`MPI_GROUP_TRANSLATE_RANKS (group, 1, rankall,
     &                               groupworld, pnum, ierror)´)
      end if
´)
```


Figure 7: Macro to send a trapezoidal submatrix

```
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, work1, work2, datatype,
     &                           mtype, ierror)´)
      MPI_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

```
define(xGSUM2D,`
      subroutine $1GSUM2D (icontxt, scope, top, m, n, a, lda,
     &                     rroot, croot)
      implicit    none
      integer     icontxt, m, n, lda, rroot, croot
      $2     a(lda,*), work(M)
      character*1 scope, top

      integer  nprow, npcol, datatype, optype, ierror
      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
        MLACS_REDUCE
      else if (rroot .eq. -1 .or. croot .eq. -1) then
        MLACS_ALLREDUCE
      else
        ierror = -1
      end if
      MLACS_DEBUG(ierror, `.ne.´, 0, $1GSUM2D_ERR)
      return
      end
´)

define(MLACS_REDUCE,`
      if (icontxt .eq. MPI_COMM_NULL) return
      INITIALIZE_COLLECTIVE(icontxt, scope, rroot, croot, rank, comm, ierror)

      MPI_CALL(`MPI_COMM_RANK(comm, locid, ierror)´)
      do i=1, n
        MPI_CALL(`MPI_REDUCE(a(1,i),work,m,datatype,optype,rank,comm,ierror)´)
        if (locid .eq. rank) then
          do j=1, m
            a(j,i) = work(j)
          end do
        endif
      end do
´)

xGSUM2D(`I´, `integer´, `MPI_INTEGER´)
xGSUM2D(`S´, `real´, `MPI_REAL´)
xGSUM2D(`D´, `double precision´, `MPI_DOUBLE_PRECISION´)
xGSUM2D(`C´, `complex´, `MPI_COMPLEX´)
xGSUM2D(`Z´, `double complex´, `MPI_DOUBLE_COMPLEX´)
```


Figure 9: Implementation of global MAX routine

```
define(xGAMX2D,`
      subroutine $1GAMX2D (icontxt, scope, top, m, n, a, lda, ra, ca,
     &                     rcflag, rroot, croot)
      implicit    none
      integer     icontxt, m, n, lda, rcflag, rroot, croot
      $2     a(lda,*), work(M)
      integer     ra(rcflag,*), ca(rcflag,*), iwrk(M)
      character*1 scope, top

      integer     datatype, optype, ierror
      integer     nprow, npcol, myprow, mypcol

      include "mpif.h"
      include "mlacs.h"
      if(m.eq.0.or.n.eq.0) return

      datatype = $3
      optype = $4
      if (rcflag .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 (rcflag .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
      else
        ierror = -1
      end if
      MLACS_DEBUG(ierror, `.ne.´, 0, $1GAMX2D_ERR)
      return
      end
´)
define(xAMXOP,`
      SUBROUTINE $1ABSMAX( invec, inoutvec, len, type )
      $2 invec(len), inoutvec(len)
      INTEGER len, type
      INTEGER i
      do i=1, len
        if (ABS(inoutvec(i)).lt.ABS(invec(i))) inoutvec(i)=invec(i)
      end do
      return
      end
´)
```

Figure 10: MLACS_REDUCE_LOC **macro for combine routines**

```
define(MLACS_REDUCE_LOC,`
      if (icontxt .eq. MPI_COMM_NULL) return
      INITIALIZE_COLLECTIVE(icontxt, scope, rroot, croot, rank, comm, ierror)

      MPI_CALL(`MPI_COMM_RANK(comm, locid, ierror)´)
      do i = 1, n
        MPI_CALL(`MPI_ALLREDUCE(a(1,i), work, m, datatype, optype,
     &                          comm, ierror)´)
        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(1,i), iwrk, m, MPI_INTEGER, MPI_MAX,
     &                       rank, comm, ierror)´)
        do j = 1, m
          if (locid .eq. rank) then
            ra(j,i) = iwrk(j)
          end if
        end do
        MPI_CALL(`MPI_REDUCE(ca(1,i), iwrk, m, MPI_INTEGER, MPI_MAX,
     &                       rank, comm, ierror)´)
        do j = 1, m
          if (locid .eq. rank) then
            ca(j,i) = iwrk(j)
          end if
        end do
      end do
´)
```


1995

TR-95-01 C. CLÉMENÇON, J. FRITSCHER, M. MEEHAN, AND R. RÜHL: An Implementation of Race Detection and Deterministic Replay with MPI. (January 1995)

TR-95-02 K. DECKER AND S. FOCARDI: Technology Overview: A Report on Data Mining. (February 1994)

TR-95-03 C. CLÉMENÇON, K. DECKER, V. DESHPANDE, A. ENDO, J. FRITSCHER, N. MASUDA, A. MÜLLER, R. RÜHL, W. SAWYER, B. J. N. WYLIE, AND F. ZIMMERMANN: Tool-Supported Development of Parallel Application Kernels. (April 1995)

TR-95-04 Y. SEO, T. KAMACHI, K. SUEHIRO, M. TAMURA (NEC CENTRAL RESEARCH LAB., KAWASAKI, TOKYO) AND A. MÜLLER, R. RÜHL (CSCS): Kemari: a Portable HPF System for Distributed Memory Parallel Machines. (June 1995)

TR-95-05 A. ENDO AND B. J. N. WYLIE: Annai/PMA Instrumentation Intrusion Management of Parallel Program Profiling. (November 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)

TR-95-07 M. GUGGISBERG, I. PONTIGGIA AND U. MEYER: Parallel Fractal Image Compression Using Iterated Function Systems. (May 1995)

1996

TR-96-01 W. P. PETERSEN: A General Implicit Splitting for Stabilizing Numerical Simulations of Langevin Equations. (February 1996)

TR-96-02 C. CLÉMENÇON, K. M. DECKER, V. R. DESHPANDE, A. ENDO, J. FRITSCHER, P. A. R. LORENZO, N. MASUDA, A. MÜLLER, R. RÜHL, W. SAWYER, B. J. N. WYLIE, F. ZIMMERMANN: Tools-supported HPF and MPI Parallelization of the NAS Parallel Benchmarks. (March 1996)

TR-96-03 BRIAN J. N. WYLIE AND AKIYOSHI ENDO: Annai/PMA Multi-level Hierarchical Parallel Program Performance Engineering. (April 1996)

TR-96-04 CHRISTIAN CLÉMENÇON, AKIYOSHI 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-96-05 M. ROTH: A Visualization System for Turbomachinery Flow. (April 1996)

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

TR-96-07 TETSUYA TAKAISHI: 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-96-09 MARC C. HOHENADEL AND PASCAL PAGNY: X-OpenWave User's Manual. (May 1996)

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

CSCS/SCSC — Via Cantonale — CH-6928 Manno — Switzerland
Tel: +41 (91) 610 8211 — Fax: +41 (91) 610 8282

CSCS/SCSC — ETH Zentrum, RZ — CH-8092 Zürich — Switzerland
Tel: +41 (1) 632 5574 — Fax: +41 (1) 632 1104

CSCS/SCSC WWW Server: http://www.cscs.ch/