Installing and Testing the BLACSv1.1 *

R. Clint Whaley †

May 5, 1997

Contents

1 Introduction **1**

2 Installation **1**

2.1 Directory Structure 1
2.2 Downloading the files 2
2.3 Unpacking 3
2.4 Editing Bmake.inc 3
2.4.1 Bmake's Section 1 4
2.4.2 Bmake's Section 2: All versions 4
2.4.3 Bmake's Section 2: PVM specific issues 5
2.4.4 Bmake's Section 2: MPI specific issues 5
2.4.5 Bmake's Section 3 6
2.5 Installation help: the INSTALL directory 6
2.5.1 MPI specific routines 7
2.6 Compiling the BLACS 7
2.6.1 Explanation of the files 7
2.6.2 Compiling the BLACS 8
2.7 Compiling the BLACS tester 8
2.7.1 Explanation of the files 8
2.7.2 Customizing the tester 8
2.7.3 Compiling the tester 9

3 Running the tester **9**

3.1 Selecting tests to run 10
3.2 SDRV tests 11
3.3 BSBR tests 12
3.4 COMB tests 13
3.5 Auxiliary tests 15

---

*This work was supported in part by DARPA and ARO under contract number DAAL03-91-C-0047, and in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC05-84OR21400.

†Dept. of Computer Sciences, Univ. of TN, Knoxville, TN 37996, `rwhaley@cs.utk.edu`

4 Understanding tester output **15**
4.1 General output 15
4.2 Error reports 16

5 Conclusions **16**

REFERENCES **18**

1 Introduction

This report covers the installation and testing of the BLACS [3]. The sections on BLACS installation will usually apply only to the BLACS obtained from netlib. The BLACS tester, however, should be run on any version of the BLACS in order to verify that they are working correctly.

There are now several vendors supporting BLACS implementations on their machines. With the BLACS being produced by many different groups, it becomes more important than ever to ensure that all versions are both syntactically and semantically correct. The BLACS tester has been written to perform at least some of these checks.

This tester calls every standard BLACS routine. Thus a successful link ensures that all standard routines at least exist in the BLACS implementation being tested. The point to point, broadcast, and combine routines may be tested as extensively as the user desires using input files. The remaining routines are lumped into the "auxiliary" tests. More information on these various tests are given in the relevant sections.

The outline for installing and testing the BLACS is given below. The following sections expand on this outline.

1. Download the BLACS, their tester, and the related papers (see Sections 2.2-2.3 for details).
2. Select a Bmake.inc example from the `BLACS/BMAKES` directory to serve as your starting point for a `Bmake.inc`, and copy it to `BLACS/Bmake.inc`. For example, if you are compiling the PVMBLACS on an alpha machine, from the `BLACS/` directory you would type `cp BMAKES/Bmake.PVM-ALPHA Bmake.inc` . (see Sections 2.1 and 2.4 for details).
3. Edit this file to fit your system (see Section 2.4 for details).
4. Compile the BLACS (see Section 2.6 for details).
5. Compile the BLACS tester (see Section 2.7 for details).
6. Test the BLACS (see Section 3 for details).

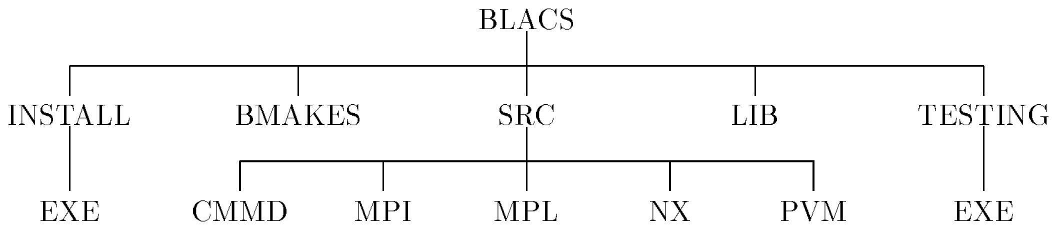
NOTE

The CMMDBLACS are discussed throughout this document. These are BLACS written for Thinking Machine's CM-5 machine. It appears that these BLACS are no longer used, so they do not appear in this release. If you have need of the CMMDBLACS, send mail to blacs@cs.utk.edu and we may make them available again.

2 Installation

2.1 Directory Structure

By default, the following structure is assumed for the BLACS and their tester:



If you have downloaded only the tester, you will receive the `BLACS/BMAKES`, `BLACS/INSTALL`, and `BLACS/TESTING` directories. If you download only the BLACS, you will receive the `BLACS/BMAKES`, `BLACS/INSTALL`, `BLACS/SRC` and `BLACS/LIB` directories. If both are downloaded, the top level directory tree will be complete.

The `BLACS/BMAKES` directory contains several example `Bmake.inc` files, which may be of help configuring the makefiles to your system. `Bmake.inc` is included by all BLACS makefiles; more details about this file are given in Section 2.4. Having an example `Bmake.inc` does not mean you can avoid examining the `Bmake.inc`. For instance, if you are using PVM on a HP machine, you can copy `BLACS/BMAKES/Bmake.PVM-HPPA` to `BLACS/Bmake.inc` and have most of the work done for you. However, you may still need to modify the `Bmake.inc` to adapt it to your system and needs; for instance you may have compiled your BLACS at a different location than is standard. Conversely, the lack of an example `Bmake.inc` in no way implies that the BLACS will not run on your system.

The `BLACS/INSTALL` directory contains several small programs which are helpful when the BLACS are being installed. They are of particular help in configuring the `Bmake.inc` file. See section 2.5 for details.

The SRC directory contains the BLACS source codes. The subdirectories under SRC indicate the various message passing libraries upon which the BLACS are presently supported. CMMD is the message passing library for Thinking Machine's CM-5, MPL runs on IBM's SP series, and NX runs on the Intel family. PVM [5] and MPI [4] run across a wide variety of UNIX and supercomputer platforms.

The BLACS libraries will be built into `BLACS/LIB`.

The `BLACS/TESTING` directory contains the tester and its related files. On all systems except PVM, the executable will be compiled into the `BLACS/TESTING/EXE` directory. On PVM, the executable defaults to `$(HOME)/pvm3/bin/<PLAT>`. The `BLACS/TESTING` directory also includes sample input files for the tester.

2.2 Downloading the files

The BLACS files which can be downloaded are:

FILE	Contents
`blacstester.tar.gz`	The BLACS tester
`cmmdblacs.tar.gz`	BLACS for Thinking Machine's CM-5
`mplblacs.tar.gz`	BLACS for IBM's SP series
`nxblacs.tar.gz`	BLACS for Intel's ipsc2, i860, delta and paragon
`pvmblacs.tar.gz`	BLACS for PVM
`mpiblacs.tar.gz`	alpha version of the BLACS for MPI
`blacs_ug.ps`	The BLACS user's guide
`blacs_install.ps`	This manual
`mpiblacs_issues.ps`	Discussion of some of the outstanding issues in the alpha release of the MPIBLACS
`mpi_prop.ps`	Some discussion of how the issues raised in mpiblacs_issues.ps might be handled by MPI
`cblacsqref.ps`	BLACS C interface quick reference guide
`f77blacsqref.ps`	BLACS f77 interface quick reference guide

If you possess a world wide web browser the BLACS and their tester may be downloaded by accessing the URL:
`http://www.netlib.org/blacs/Blacs.html`.

Downloading by anonymous ftp can be accomplished by `ftp ftp.netlib.org` directory `blacs/`.

2.3 Unpacking

The tar files for the tester and the BLACS will create their portions of the previously discussed directory tree. They create a directory `BLACS` in the directory where they are unpacked. Subsequent unpacks should be performed in this same directory (i.e., above the `BLACS` directory).

The unpacking of the compressed tar file may be accomplished by `gunzip -c FILE | tar xvf -`. Note that gnu tar was used to create these tar files. We have reports that SUN4's tar may issue a error message when untarring, but have no reports of this actually causing problems.

2.4 Editing Bmake.inc

The first step is to modify the BLACS make include file `Bmake.inc` to match your system. This file sets up various macros needed for make and the BLACS. `Bmake.inc` is roughly split into three sections. Bmake's section 1 defines the macros necessary to find/name the various libraries and executables. Bmake's section 2 sets up internals in the BLACS, allowing the user to perform fine-tuning for a platform, change how the BLACS behave, etc. Bmake's section 3 defines macros dealing with compilers, linker/loaders, etc.

The directory `BLACS/BMAKES` contains examples of `Bmake.inc`'s for various systems. This, together with the internal comments and help from the INSTALL directory, should enable the user to create a usable `BLACS/Bmake.inc` for his system. The format for the names of the example `Bmake.inc` files is:

`Bmake.<COMMLIB>-<PLAT>`, where `<COMMLIB>` indicates the communication library, which will be one of `CMMD`, `MPI`, `MPL`, `NX`, and `PVM`. `PLAT` represents the platform or architecture where the message passing library will run. Examples include `I860` (Intel i860), SUN4, HPPA, etc. So, if you wish to run the MPIBLACS on an IBM SP2, you would choose the file `BLACS/BMAKES/Bmake.MPI-SP2` as your starting point for a `BLACS/Bmake.inc`.

2.4.1 Bmake's Section 1

In Bmake's section 1 we declare where our libraries are, where the executables should be placed, etc. The use of most of these macros should be apparent from reading the comments. We will briefly mention here some of the more obtuse macros.

The first of these is `BLACSDBGLVL`. This macro controls the debug level the BLACS are compiled with. At present, the BLACS possess only two levels of debug: `0` and `1`. The example `Bmake.inc` files default to debug level 0. At this level, almost no error checking is done by the BLACS. Incorrect parameters will as a rule not be caught, and will often generate non-deterministic behavior. This level of debug is most useful once a code is in production mode with all bugs ironed out, where performance is the main issue.

Until all of the user's code has been thoroughly tested, it is recommended that he compile and link to the level 1 debug BLACS. This level of debug is non-intrusive performance-wise: in the main no off-process access of debug information is required. The main benefit of this mode is better parameter checking, which we have found to be very useful when developing code.

2.4.2 Bmake's Section 2: All versions

Bmake's section 2 sets C preprocessor values for the BLACS. If the makefile is not used, these options may also be varied by editing the file `Bconfig.h`. Many of these options are `COMMLIB` dependent, and are thus discussed below.

SYSINC The first standard entry in Bmake's section 2 is SYSINC. This variable sets up the search path for system specific include files. For example, this will tell the compiler where to find `mpi.h` for the MPIBLACS.

INTFACE Indicates what namespace interface is required to have Fortran77 call a C routine. If you are unsure of how to set this variable, run the routine `xintface` as described in section 2.5.

DEFBSTOP, DEFCOMBTOP These macros allow the user to vary what topology the BLACS default topology (`TOP = ' '`) actually calls when the user does a broadcast (`DEFBSTOP` should be set) or a combine (`DEFCOMBTOP` should be set). Usually the defaults built into the BLACS will be fine. An example of when the user would find these macros useful can be found in the MPIBLACS. The space topology will call MPI's built-in broadcast function. On some systems, with some implementations of MPI, this can be less efficient than, for instance, using `TOP = '1'`. If the user has determined this, he could set, for instance, `DEFBSTOP = -DDefBSTop="'1'"`.

2.4.3 Bmake's Section 2: PVM specific issues

CATCHOUT Users may be confused by this option. PVM allows the user to have program output sent to the standard out of the spawning process, or to have the output go to the file `/tmp/pvml.<userid>`. By default the BLACS send the output to the standard out of the spawning process. If the user specifies `CATCHOUT = -DBLACSNoCatchout`, the BLACS instead send the output to the `/tmp` files.

Sending processes' output to the standard out of the spawning process, requires that the spawning process service the messages containing the printing information. This may not be possible in some error conditions, making it necessary to use the `/tmp` files to see all of the messages.

2.4.4 Bmake's Section 2: MPI specific issues

Please note that all example MPI Bmake.inc files set these MPI-specific flags to the values used by MPICH[1, 2]. This is simply because this is the version of MPI that the BLACS have been most widely tested on.

SENDIS If MPI_Send is locally-blocking on your system (see [3] for details on blocking), you may increase the efficiency of your BLACS by setting this macro to `-DSndIsLocBlk`. If this macro is left blank, the BLACS assume MPI_Send is globally-blocking, and buffering combined with non-blocking sends will be used to make the BLACS point to point send locally-blocking.

BUFF By default the BLACS use MPI's datatype support for sending/receiving of non-contiguous messages. On some MPI implementations, this can result in inefficient code, especially when broadcasting or combining using a topology other than the default. In particular, on those systems where datatypes are supported by buffering rather than sending the non-contiguous message, this may be inefficient. You can tell the BLACS to do their own packing by setting `BUFF = -DNoMpiBuff`.

TRANSCOMM As discussed in [6] the BLACS will need to translate between Fortran77 and C communicators. In particular, if the BLACS are internally using the C interface to MPI, and the user calls `BLACS_GRIDMAP` or `BLACS_GRIDINIT` from Fortran77, or if these routines are called from C, and the BLACS are internally calling the Fortran77 interface, the passed in MPI communicator will need to be translated to the other language.

If this macro is left blank, the BLACS do this translation by translating all ranks to `MPI_COMM_WORLD`, and thus forming an identical context in the other language (it is assumed that `MPI_COMM_WORLD` is the same in both languages). This is, in general, an unsatifying solution, as it causes all processes to block on each call to grid formation where translation must occur. This problem is discussed in [6].

This translation can be made more efficient if the user knows something about his system. If the C and Fortran77 contexts are the same, the BLACS can be told no translation need be done by setting `TRANSCOMM = -DCSameF77`.

If you are using MPICH or one of its close derivatives, the BLACS should be able to use MPICH internals to do the translation. To signal this, set `TRANSCOMM = -DUseMpich`. If your system's pointers are longer than its integers, you'll need to set MPICH's variable indicating this. We would then have `TRANSCOMM = -DUseMpich -DPOINTER_64_BITS=1`. This variable is set in the MPICH installation. If you don't have access to your MPICH installation, run the routine `xtranscomm` as described in Section 2.5 to determine whether POINTER_64_BITS=1 should be set or not.

WHATMPI Almost all MPI calls in the BLACS utilize macros so that at compile time the user can select whether the BLACS internally call the Fortran77 interface to MPI, or the C interface to MPI. Some MPI calls (mainly those used to translate a communicator from one interface to another) explicitly call one interface or another, and are thus unaffected by changing this macro. Generally, it will suffice to leave this macro blank, leaving the BLACS to choose which to use, unless the user has some strong reason to prefer one interface to another.

SYSERRORS Some incorrect MPI implementations cannot handle zero byte data types. If this is the case with your MPI library, set `SYSERRORS = -DZeroByteTypeBug`. If you are unsure about how to set this macro, run the routines `xtc_CsameF77` and `xtc_UseMpich` as described in Section 2.5.

2.4.5 Bmake's Section 3

Bmake's section 3 of `Bmake.inc` is probably the most straightforward. Here we specify what compilers, linkers, etc., the BLACS and their tester should use for compilation. The comments in this section of Bmake should be sufficient for the user to make any necessary modifications.

2.5 Installation help: the INSTALL directory

This directory contains several small routines which should help a user in installing the BLACS. All of these routines should be compiled and ran on the platform for which the BLACS are being installed (for instance, if the user is running the MPIBLACS on top of MPICH, he would use `mpirun` to execute these routines). Note that the user should set up sections 1 and 3 of `Bmake.inc` before compiling these routines. On all systems except PVM, the executables for these routines will default to the `BLACS/INSTALL/EXE` directory. On PVM, they will by default be placed in `$(HOME)/pvm3/bin/<PLAT>`.

size.f This routine can be compiled by `make xsize`. The resulting executable will tell the user the correct settings for the size variables in `btprim_PVM.f`'s `btsizeof` routine. This routine can be run with any BLACS, but is generally only needed for the PVMBLACS.

Fintface.f Cintface.c These routines can be compiled by `make xintface`. The resulting executable will give the proper setting for `Bmake.inc`'s INTFACE macro. This routine can be run with any BLACS.

2.5.1 MPI specific routines

syserrors.c This routine can be compiled by `make xsyserrors`. If a run of the resulting executable completes, `SYSERRORS` should be left blank. Otherwise, the user should set it to `-DZeroByteTypeBug`.

tc_fCsameF77.f tc_cCsameF77.c These routines can be compiled by `make xtc_CsameF77`. The resulting executable will indicate whether it is OK to set TRANSCOMM to `-DCsameF77`. If it is not, check if your MPI has the MPICH translation routines by running `xtc_UseMpich`.

tc_UseMpich.c This routine can be compiled by `make xtc_UseMpich`. Compile and run `xtc_CsameF77` first. If it does not indicate it is safe to set TRANSCOMM to `-DCsameF77`, compile and run this routine. If the link fails, or if the routine fails to complete, leave TRANSCOMM blank.

cmpi_sane.c This routine can be compiled by `make xcmpi_sane`. It calls the C interface to MPI. Many BLACS questions really turn out to be MPI questions. This is an extremely simple MPI routine which allows the user to verify the the most basic MPI usage is working on his platform. When the user is unable to get the BLACS tester to even start running under his MPI, it is advisable to run this routine and verify that MPI is working, and that the user is issuing the correct command to run MPI on his platform. This routine has a cpp macro `WASTE_SIZE` which can be increased/decreased to give the program arbitrary memory usage (memory usage can be important in determining whether or not apawning of processes succeeds).

fmpi_sane.f This routine can be compiled by `make xfmpi_sane`. It calls the Fortran77 interface to MPI, and exists to do the same checks for the Fortran77 interface that `cmpi_sane.c` provides for the C interface. It has a parameter for varying memory usage as well, called `WASTESZ`.

2.6 Compiling the BLACS

2.6.1 Explanation of the files

All `BLACS/SRC/<COMMLIB>` directories have a subdirectory called `INTERNAL`. Routines in this directory are internal to the BLACS, and are thus not designed to be called by the user directly. Routines in the `BLACS/SRC/<COMMLIB>` directory are user-callable routines, and thus have both C and Fortran77 interfaces. Note that all standard non-communication routines have `blacs_` prefixed to their names. This distinguishes them from the service routines provided by the library but not guaranteed by the standard, such as `dcputime00` and `kbsid`.

The MPIBLACS has a further standardization of naming schemes. All BLACS internal routines are prefixed by `BI_`, as are all global variables. This standardization is provided to help minimize name-space conflicts with the user's libraries. We anticipate that the next release of the BLACS will extend this idea to all BLACS versions.

2.6.2 Compiling the BLACS

Make sure you have the correct `Bmake.inc` for your platform. To compile the BLACS, simply go to the `BLACS/` directory, and type `make <COMMLIB>`. For instance, `make pvm` compiles the PVMBLACS, `make mpi` compiles the MPIBLACS, etc. To remove the object files after compilation, simply type `make <COMMLIB> what=clean`, eg., `make mpi what=clean`.

NOTE: when the BLACS are are archived into library format, the archiver will probably report that it is truncating some long file names. *This is not an error*, and should not cause any problems.

2.7 Compiling the BLACS tester

2.7.1 Explanation of the files

If you do a directory listing in BLACS/TESTING, you will find the following files:

- `blacstest.f` This file contains most system-independent code for the BLACS tester.
- `tools.f` Some tool routines used by the tester. All of these routines come from LAPACK. They may also be found in ScaLAPACK's TOOLS directory.
- `btprim_CMMD.f` This file provides `blacstest.f` with the system primitives it needs to run on Thinking Machine's CMMD message passing layer. Thus it will be used to test the BLACS on the CM-5.
- `btprim_MPI.f` This file provides `blacstest.f` with the system primitives it needs to run on MPI.
- `btprim_MPL.f` This file provides `blacstest.f` with the system primitives it needs to run on IBM's MPL message passing layer. Thus it will be used to test the BLACS on the SP series.
- `btprim_NX.f` This file provides `blacstest.f` with the system primitives it needs to run on Intel's NX message passing layer. Thus it will be used to test the BLACS on such machines as Intel's iPSC/2, iPSC/860, Delta, and Paragon.
- `btprim_PVM.f` This file provides `blacstest.f` with the system primitives it needs to run on PVM.
- `Makefile` The tester's makefile.

2.7.2 Customizing the tester

The primary customization of the tester will involve setting the parameter `MEMELTS` in `blacstest.f`. This parameter controls the number of double precision elements in the tester's main array. The tester will section this array up as needed for all non-local arrays. This value must be set large enough to run the largest matrix test, and small enough so that the tester fits into memory. Most users will be satisfied with the supplied default value.

For various PVM platforms In order to section the main array, the tester needs to know the size, in bytes, of each data type. In `btprim_PVM.f` there is a routine called `btsizeof` which returns this information. You should make sure the values returned by this routine are correct for your platform. If you are unsure what values are correct for your system, run the routine `xsize` as described in Section 2.5.

For unsupported platforms If the user wishes to run the tester on a platform which is not presently supported, it will be necessary to create a blacstester primitive file for that platform. This should be relatively easy to do by simply substituting calls in one of the supported primitive files. The six routines in the primitive file are straightforward and well documented enough that it should be easy to write them for the desired message passing library.

The presently supported tester primitive files are: `btprim_CMMD.f, btprim_MPI.f, btprim_MPL.f, btprim_NX.f, btprim_PVM.f`, as discussed earlier. Often one of these will be very close to what you need for a new platform. For instance, the MPL version was produced by making minor modifications to the CMMD file.

2.7.3 Compiling the tester

The first step is to modify the BLACS make include file `Bmake.inc` to match your system. Section 2.4 explains this in detail.

Once this is done, compilation is accomplished by moving into the `BLACS/TESTING` directory, and typing `make`. By default, the a tester is built for both the C and Fortran77 interfaces. Note that the C interface BLACS are called via a series of wrappers with the same name-space as the Fortran77 interface BLACS. Therefore, if your BLACS implement either interface as wrappers around the other, you will not be able to test one interface explicitly, as there would be name-space conflicts. The BLACS discussed in this paper implement each interface seperately, so this is not a problem.

There is also a top level makefile which may be used instead. In the `BLACS/` directory, typing `make` with no arguments gives help. To compile the tester using this makefile, simply type `make tester`. To remove object files, type `make tester what=clean`.

3 Running the tester

On all systems except PVM, the tester executable and input files will default to the `BLACS/TESTING/EXE` directory. The name of the executables on such systems will be `x[F,C]btest_<COMMLIB>-<PLAT>-DEBUGLEVEL` (e.g. `xFbtest_MPI-SUN4-0`), where F indicates the Fortran interface is being called, and C indicates the C interface is being tested. On PVM, they will by default be placed in `$(HOME)/pvm3/bin/<PLAT>`, and since the platform information is encoded in the path, the name format is: `x[F,C]btest_<COMMLIB>-DEBUGLEVEL` (eg. `xCbtest_PVM-0`). The actual method by which the executable is run varies widely amongst systems. See your local system guide for details on running parallel programs for your system. Note that the example input files provided with the tester require a minimum of 4 processes to run.

3.1 Selecting tests to run

The overall behavior of the tester is controlled by the input file `bt.dat`. An example of a legal `bt.dat` is:

```
'Sample BLACS tester run'      Comment line
6                              device out
'blacstest.out'                output fname
'T'                            Run SDRV?
'T'                            Run BSBR?
'T'                            Run COMB?
'T'                            Run AUX?
5                              Number of precisions
'I' 'S' 'D' 'C' 'Z'            Values for precision
0                              Verbosity level
```

The first line is a comment line, which will be regurgitated by the tester as the first line of output. The second line is the device number to use for output. If the device number is anything but 0 or 6 (standard error and standard out, for most systems), the tester writes all output to a file, whose name is given in line 3.

Lines 4 through 7 indicate whether that form of test should be performed. If the input is 'T', the test is performed. If it is 'F', the tests are not performed. The comments (which are, of course, optional) in the input file use some abbreviations found throughout the tester. These abbreviations fall in line with those used in the BLACS, and so should be familiar to the user. They are:

- **SDRV**: Point to point send/receive tests,
- **BSBR**: Broadcast tests,
- **COMB**: Combine tests, which are further specified by:
  - **AMX**: Absolute value element-wise maximization,
  - **AMN**: Absolute value element-wise minimization,
  - **SUM**: Element-wise summation.
- **AUX**: Auxiliary tests, which handle the rest of the BLACS routines.

Line 8 indicates the number of data types to test. There are 5 data types supported by the BLACS, and they are selected on line 9. The possible values are:

Initial	Data type
I	Integer
S	Single precision real
D	Double precision real
C	Single precision complex
Z	Double precision complex

The final line indicates the *verbosity* level. This is an easy way for the user to vary the amount of output he gets from the tester. At present, there are 3 levels:

0. A message is printed when a class of tests are begun (for instance 'INTEGER SDRV TESTS: BEGIN', meaning integer send/receive tests have started). When the class is finished, a message is printed out telling if the tests passed or failed. If there were failures, the number of failed tests is given.
1. In addition to the output associated with level 0, a header is printed out before each class of tests. This header indicates which tests will be run during this test class. It amounts to a printing of the relevant input file.
2. A message is printed at the beginning and ending of each individual test, as well as the output done by previous verbosity levels.

Regardless of the verbosity level, errors always result in messages being printed.

3.2 SDRV tests

The first class of tests in the BLACS is SDRV. This stands for point to point send and receive, and corresponds to calls to the BLACS routines `□GESD2D`/`□TRSD2D` and `□GERV2D`/`□TRRV2D`. The input file for this class of tests is `sdrv.dat`. The user should modify this file to perform the specific tests he requires. An example is:

```
5                        Number of shapes
'U' 'U' 'L' 'L' 'G'      UPLO
'U' 'N' 'U' 'N' 'E'      DIAG
3                        Number of matrices
1 25 13                  M
7 19 32                  N
3 25 14                  LDASRC
2 25 22                  LDADEST
4                        Number of src/dest pairs
0 1 3 0                  RSRC
0 0 0 2                  CSRC
0 1 2 0                  RDEST
1 1 0 0                  CDEST
3                        Number of grids
2 4 1                    NPROW
2 1 4                    NPCOL
```

In general, the tests to be run are indicated by blocks in the input file. The first line of such a block indicates the number of values of a quantity which will be tested, and subsequent line(s) of the block give the separate values. In the above example, such a block is lines 1 through 3, which control the shape the matrix will possess.

The total number of tests which will be run can be calculated by multiplying the first lines of each block. In the above input file, we therefore see there will be 180 tests attempted.

If the user specifies a process coordinate for source or destination which is invalid for a particular process grid being tested, that test will be skipped.

The values controlled by `sdrv.dat` are:

- **UPLO** indicates if the matrix is upper triangular (`UPLO = 'U'`), lower triangular (`UPLO = 'L'`), or general rectangular (`UPLO = 'G'`).
- **DIAG** specifies if the diagonal of a triangular matrix needs to be communicated. If `DIAG = 'U'` (unit diagonal), the diagonal is not communicated. If the diagonal is to be communicated, `DIAG = 'N'` (non-unit diagonal) should be selected. If the matrix is general rectangular, `DIAG` will be ignored.
- **M** $M \geq 0$.
  The number of rows in the matrix,
- **N** $N \geq 0$.
  The number of columns in the matrix,
- **LDASRC** $LDASRC \geq M$.
  The leading dimension of the matrix on the source processor.
- **LDADEST** $LDADEST \geq M$.
  The leading dimension of the matrix on the destination processor.
- **RSRC** $0 \geq RSRC < NPROW$.
  The process row of the source (the sender) of the message.
- **CSRC** $0 \leq CSRC < NPCOL$.
  The process column of the source (the sender) of the message.
- **RDEST** $0 \leq RDEST < NPROW$.
  The process row of the destination of the message.
- **CDEST** $0 \leq CDEST < NPCOL$.
  The process column of the destination of the message.
- **NPROW** $NPROW \geq 1$.
  The number of rows in process grid.
- **NPCOL** $NPCOL \geq 1$.
  The number of columns in process grid.

3.3 BSBR tests

The second class of tests in the BLACS is BSBR. This stands for broadcast/send and broadcast/receive, and corresponds to calls to the BLACS routines `□GEBS2D/□TRBS2D` and `□GEBR2D/□TRBR2D`. The input file for this class of tests is `bsbr.dat`, and an example is:

```
3                                 Number of scopes
'R' 'C' 'A'                       Scopes
8                                 Number of topologies
'I' 'S' 'H' '1' 'd' 'm' '4' ' '   TOP
5                                 Number of shapes
'G' 'U' 'U' 'L' 'L'               UPLO
'E' 'U' 'N' 'U' 'N'               DIAG
3                                 Number of matrices
1 25 13                           M
7 19 32                           N
3 25 14                           LDASRC
2 25 22                           LDADEST
4                                 Number of sources
0 1 3 2                           RSRC
0 0 1 1                           CSRC
4                                 Number of grids
2 4 1 4                           NPROW
2 1 3 1                           NPCOL
```

Most of these parameters have been explained in Section 3.2. We note the following: this input file does not have lines for RDEST and CDEST, because a broadcast means that everyone in the scope except the source is a destination process. We also add two new quantities:

- **SCOPE** The scope of the broadcast:
  - **'R'**: Process row `RSRC` participates in broadcast.
  - **'C'**: Process column `CSRC` participates in broadcast.
  - **'A'**: Entire process grid participates in broadcast.
- **TOP** The BLACS topology to be used in the broadcast.

There are some special case topologies that result in atypical test behavior. `TOP = 'M'` and `TOP = 'T'` are topologies which behave differently based on calls to `BLACS_SET`. Therefore, in order to test these topologies, a single "test" is actually a series of tests, which loops over all relevant calls to `BLACS_SET`.

3.4 COMB tests

The third class of tests in the BLACS is COMB. This stands for combine tests, and corresponds to calls to the BLACS routines `□GSUM2D`, `□GAMX2D`, and `□GAMN2D`. The input file for this class of tests is `comb.dat`, and an example is:

```
3                  Number of OPs
'+' '>' '<'        Combine operations to perform
3                  Number of scopes
'R' 'C' 'A'        values for scopes
```

```
2                        Repeatability flag (0=no-rep, 1=rep, 2=both)
2                        Coherence flag (0=no-coh, 1=coh, 2=both)
6                        Number of topologies
' ' '1' '2' 'H' '3' 'f' TOP
5                        Number of matrices
3 1 2  25 13             M
5 1 3  19 32             N
5 1 4  25 14             LDASRC
9 1 5  25 22             LDADEST
4 1 -1 25 22             RCFLAG
4                        Number of dests
0 -1 0 2                 RDEST
0 -1 1 0                 CDEST
4                        Number of grids
2 1 3 4                  NPROW
2 4 1 1                  NPCOL
```

Again, most of these parameters have been explained in previous sections. Note that RSRC and CSRC are not supplied; like in the broadcast, they are implied by RDEST, CDEST, and the scope. The new quantities are:

- **OP** The combine operation to perform:
  - '+': Test summation combine,
  - '>': Test absolute value maximization combine,
  - '<': Test absolute value minimization combine.
- **Repeatability flag** The BLACS allow the user to specify whether topologies should be forced to be repeatable (see [3] for and explanation of repeatability) or not. This flag may have 3 values:
  1. : Each combine test is called with the repeatability flag not set.
  2. : Each combine test is called with the repeatability flag set.
  3. : Each combine test is two calls, one with the repeatability flag not set, and one with it set.
- **Coherence flag** The BLACS allow the user to specify whether topologies should be forced to be coherent (see [3] for and explanation of coherence) or not. This flag may have 3 values:
  1. : Each combine test is called with the coherence flag not set.
  2. : Each combine test is called with the coherence flag set.
  3. : Each combine test is two calls, one with the coherence flag not set, and one with it set.
- **RCFLAG** This input is ignored for summation tests. For max/min, it is defined as in the BLACS.

Note that as in broadcast, `TOP = 'T'` will result in multiple tests for each "test" reported in output.

3.5 Auxiliary tests

The final series of tests is referred to as the AUX tests. Unlike the other categories, this group has no input file. Still, some knowledge about its operation should prove helpful to the user.

Many of the auxiliary routines are hard to test. In some cases, they cannot be tested at all. In others, they could only be tested if the test were to produce a hang. For those routines that we can test to some appreciable degree, the following message pair will be generated:

```
RUNNING <test type> TEST
PASSED/FAILED <test type> TEST
```

For the routines which we are unable to test, we just call them. This makes sure that the routine exists, and at least has no gross defects. The message pair will then be:

```
CALL <routine>
DONE <routine>
```

An example of a routine for which we have no good test is BLACS_BARRIER. It should hold up the execution of all processes within the specified scope, until they have all called the routine. The only test we have devised for this is to have a given process fail to call the routine, and make sure that no process exits the BLACS_BARRIER call. Since this would result in the tester hanging when the BLACS being tested are working correctly, we do not perform the test.

The final auxiliary test checks if BLACS_ABORT is working correctly. If it is, your processes will be killed. Since the machine is killed you may not get a message indicating the test has passed. *This is not an error.*

In general, it is advised to run the auxiliary tests only once for a particular number of processors. This avoids having the tester kill the machine each run. Since the auxiliary tests do not base their tests upon an input file, no new insight is gained by additional testing runs.

4 Understanding tester output

Please note that this section deals with the output generated by the SDRV, BSBR, and COMB tests. AUX test output has been discussed in the previous section.

4.1 General output

At least two lines of output will be printed for each class of tests. Upon starting the class of tests, the following message is always generated:

```
<DATA TYPE> <TEST TYPE> TESTS: BEGIN.
```

When the class of tests are finished, one of two messages will be printed. If all tests passed, the following message is generated:

```
<DATA TYPE> <TEST TYPE> TESTS: ALL XXXXX TESTS PASSED.
```

If some tests failed or were skipped due to illegal input, the following is printed

```
<DATA TYPE> <TEST TYPE> TESTS: XXXXX TESTS; XXXX PASSED, XXXX SKIPPED, XXXX FAILED.
```

At the end of all tests, one of two messages is printed. If all tests passed, the user is so informed. Otherwise, the message will indicate that there were failures. This should alert the user he should look more carefully at the preceding output.

If verbosity is increased, additional printing is done, as previously mentioned. If the maximum verbosity is chosen (VERB=2), a summary line is printed as each individual test is begun, and then repeated with either 'PASSED' or 'FAILED' upon completion.

4.2 Error reports

There are five basic errors which will be reported:

1. Memory overwrite before beginning of matrix.
2. Memory overwrite after end of matrix.
3. Memory overwrite in LDA - M gap.
4. Memory overwrite in complementary triangle.
5. Invalid element in matrix.

The first four errors indicate that memory is being overwritten. The matrix is "padded" so that overwrites in its vicinity can be detected. The LDA - M gap refers to the area in each column of the matrix, between the last element to be referenced (M), and the column length of the fortran array (LDA).

The complentary triangle is that section of a trapezoidal matrix which remains untouched (above or below the diaginal, depending on UPLO), which distinguishes it from a rectangular matrix.

All error messages consist of two lines. The first identifies which error has been detected, and the second indicates what the tester expected to receive, and what it actually found.

5 Conclusions

If you have questions involving the BLACS or the tester, send mail to `blacs@cs.utk.edu`. It is highly recommended that you first look at the BLACS homepage, which has a troubleshooting section.

If you have found errors in the UT BLACS, send e-mail describing the error to `blacs@cs.utk.edu`. Specify the type of machines you ran on, and give the offending output. Please also include the command you used to run the tester. If you are using PVM, please include the contents, if any, of your `/tmp/pvml.<uid>` files. Brevity is appreciated, so giving the smallest run which produces the error is strongly encouraged.

Acknowledgments: The author would like to thank Martin Do and J. Michael Hammond for their help in the production of the BLACS tester.

References

[1] Patrick Bridges, Nathan Doss, William Gropp, Edward Karrels, Ewing Lusk, and Anthony Skjellum. "Users' Guide to `mpich`, a Portable Implementation of MPI", 1995. Available via world wide web from URL = `http://www.mcs.anl.gov/mpi/mpich/index.html`.

[2] Patrick Bridges, Nathan Doss, William Gropp, Edward Karrels, Ewing Lusk, and Anthony Skjellum. "Installation Guide to `mpich`, a Portable Implementation of MPI", 1995. Available via world wide web from URL = `http://www.mcs.anl.gov/mpi/mpich/index.html`.

[3] Jack Dongarra and R. Clint Whaley. "A User's Guide to the BLACS v1.1". Technical Report UT CS-95-281, LAPACK Working Note #94, University of Tennessee, 1995.

[4] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard . *International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4), 1994. Special issue on MPI. Also available electronically, the url is `ftp://www.netlib.org/mpi/mpi-report.ps`.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. The book is available electronically, the url is `ftp://www.netlib.org/pvm3/book/pvm-book.ps`.

[6] R. Clint Whaley. "Outstanding Issues in the MPIBLACS", 1995. Available on netlib from the `blacs/` directory.