Essential Introduction to the NAG Parallel Library

This document is essential reading for any prospective user of the NAG Parallel Library.

Contents

1 The Library and its Documentation

1.1 Structure of the Library

The NAG Parallel Library is a collection of parallel routines written in Fortran 77 for the solution of numerical and statistical problems targeted primarily at distributed memory machines. The word 'routine' is used to denote 'subroutine' or 'function'.

The Library is divided into **chapters**, each devoted to a branch of numerical analysis or statistics. Each chapter has a three-character name and a title, e.g.

D01 − Quadrature

The chapters and their names are consistent with those used in the NAG Fortran Library, and are loosely based on the ACM modified SHARE classification index [13].

Most documented routines in the Library have seven-character names, beginning with the characters of the chapter name, e.g.

D01AUFP

Note that the second and third characters are **digits**, not letters; e.g. 0 is the digit zero, not the letter O. The last two letters of each routine name always appear as 'FP' in the documentation.

Chapter F07 (Linear Equations (ScaLAPACK)) and Chapter F08 (Least-squares Problems (ScaLAPACK)) contain routines derived from the ScaLAPACK project. These routines have NAG-style names as well as the actual ScaLAPACK names, e.g. F07ADFP (PDGETRF). Details regarding these alternate names can be found in the relevant Chapter Introductions.

A few of the documented routines, those which are identical to routines in the NAG Fortran Library, have six-character names. The naming structure of those routines is the same as for the other routines, except for the omission of the final 'P' in the routine name. See also Section 1.10.

1.2 Structure of the Manual

The **NAG Parallel Library Manual** is the principal documentation for the NAG Parallel Library. It has the same chapter structure as the Library: each chapter of routines in the Library has a corresponding chapter (of the same name) in the Manual. The chapters occur in alphanumeric order. General introductory documents are placed at the beginning of the Manual, and the indexes follow the routine documents.

Each chapter consists of the following documents:

Chapter Contents, e.g. Chapter D01 − Quadrature;

Chapter Introduction, e.g. Introduction − D01;

Routine Documents, one for each documented routine in the chapter.

A routine document has the same name as the routine which it describes. Within each chapter, routine documents occur in alphanumeric order. Exceptionally, some chapters (X01, X02), which contain simple support routines, do not have individual routine documents; instead, all the routines are described together in the Chapter Introduction.

1.3 Parallelism and Data Distribution

The NAG Parallel Library is intended primarily for distributed memory parallel machines, including networks and clusters of machines, although it can readily be used on shared memory systems that implement MPI ([12], [8]) or PVM ([7]) and the Library can of course also be run on single processor machines. (Please contact NAG to enquire about availability of implementations, see Section 5.)

The Library supports parallelism and memory scalability and has been designed to be portable across a wide range of parallel machines. But the Library cannot be expected to achieve near peak performance for all problems on all machines and, in particular, algorithmic scalability is unlikely across the whole Library.

In this section we give some background to the design of the Library in order to aid understanding of the use of the Library.

1.3.1 Library model

The NAG Parallel Library is designed to execute on a two-dimensional grid of logical processors. The Library assumes a Single Program Multiple Data (SPMD) model of parallelism (see [11]) in which a single instance of the user's program executes on each of the logical processors with different data. In this model, during the computation phases, each logical processor computes on it its own portion of the data and during the communication phases, the processors exchange data using a message passing library (in our case BLACS and either MPI or PVM, see Section 1.3.4).

In the current version of the Library, only one Library Grid of logical processors is allowed to execute Library routines at any one time. In future versions of the Library this restriction may be removed, so that multigridding (concurrent application of a routine on multiple instances of the data) may be supported by the Library. Although only one 'Library Grid' is permitted at any one time, users may be able to define more than one grid themselves. See the Tutorial for an example.

The manual uses the terminology $\{i, j\}$ to denote the logical processor in row i and column j of the grid. The Library designates a **root processor** and this is usually processor $\{0, 0\}$, **but this is not necessarily the case**. However, the Library provides a routine which returns the row and column coordinates of the root processor within the Library Grid. See Section 1.3.4 below. (Those users who are familiar with MPI should note that there is no guaranteed one-to-one correspondance between the $n_p \times i + j$ processor and the MPI 'rank' in the Library Grid, where n_p denotes the number of columns in the Library Grid.)

Whilst the Library assumes that the logical topology (virtual topology) is a two-dimensional grid of processors, no assumption is made about the actual topology of the system.

1.3.2 Definitions

The following definitions are used in describing the Library Grid and the data distributions throughout this manual:

m_p – the number of rows in the logical processor grid
n_p – the number of columns in the logical processor grid
p – $m_p \times n_p$, the total number of processors in the logical processor grid
$\lceil x \rceil$ – the ceiling function of x, which gives the smallest integer not less than x

Further definitions are introduced where necessary in individual routine document.

1.3.3 Data distribution

For many routines the distribution of data to the logical grid of processors is the responsibility of the user, as is the collection of results if this is required. There are however routines in the Library to aid the distribution and collection.

The Library distinguishes between local and global data.

Local:
: local input arguments may have different values on each logical processor on entry to the library routine. Local output arguments may be assigned different values on different logical processors on exit from a library routine. The significance of these values is given in the routine documentation.

Global:
: global input arguments **must** have the same value on each logical processor on entry to a library routine. If this is not the case then all library routines exit with an error condition (see Section 2.3). For global input array arguments this condition applies elementwise. Global output arguments will be assigned the same value on all processors on exit from a library routine.

Section 3 of each routine document describes the local and global arguments and, where relevant, the data distribution.

The data distributions used are chosen to try to minimise data movement and to achieve reasonable load balancing and currently two data distributions associated with matrices and vectors are used by the Library.

(i) **Cyclic 2-d block distribution of matrices.** This distribution, also referred to as a block-scattered distribution, is used by the routines in Chapter F04 and the ScaLAPACK routines in Chapters F07 and F08, and is such that block rows of the matrix are distributed in wrapped fashion to the associated row of logical processors and block columns are distributed in wrapped fashion to the associated column of logical processors. Here the terms block row and block column refer to one or more contiguous rows or columns of a matrix which are treated as a single entity from the algorithmic point of view.

Figure 1 and Figure 2 illustrate the cyclic 2-d block distribution of a matrix consisting of twelve row and column blocks on a two by three grid. The block rows (and the block columns) of the matrix are numbered from 1 to 12.

	1	2	3	4	5	6	7	8	9	10	11	12
1	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
2	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
3	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
4	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
5	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
6	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
7	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
8	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
9	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
10	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
11	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
12	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}

Figure 1
Block distribution over a 2 by 3 logical grid of processors.

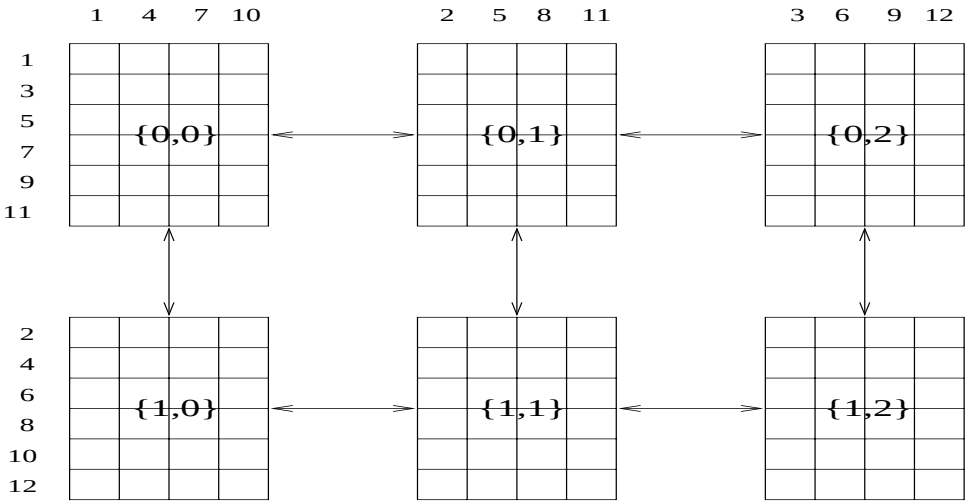

Figure 2
Data distribution from the processors' point of view.

See Chapter 4 of [10] and [4] for further information on the cyclic 2-d block distribution, including mathematical definitions.

(ii) **Block column distribution of matrices.** This distribution is used by the routines in Chapter F02 of the Library. Here, a block is a block of complete contiguous columns and is such that block i of the matrix is placed on logical processor $(i-1)$, when regarding the grid as a linear array of

processors. Thus, columns of the matrix are allocated to logical processors on the 2-d grid row by row (i.e., in row major ordering of the grid).

Each logical processor that contains columns of the matrix, contains $N_b = \lceil n/p \rceil$ columns, except the last processor that actually contains data, which may hold fewer than N_b columns. This processor will contain $\mathrm{mod}(n, N_b)$ columns if $\mathrm{mod}(n, N_b) \neq 0$, and will contain N_b columns otherwise. Some logical processors may not contain any columns of the matrix if n is not large relative to p; but if $n > (p-1)^2$, then all processors will certainly contain columns of the matrix. The number of logical processors that contain columns of the matrix, is given by $\lceil n/N_b \rceil$.

The following figure illustrates the distribution of a matrix with 41 columns on a two by four logical grid. Notice that the first six processors contain $N_b = 6$ columns, the penultimate processor contains 5 columns and the final processor does not contain any columns.

processor 0 columns (1:6)	processor 1 columns (7:12)	processor 2 columns (13:18)	processor 3 columns (19:24)
processor 4 columns (25:30)	processor 5 columns (31:36)	processor 6 columns (37:41)	processor 7

1.3.4 Message passing, context and running tasks

If you intend to use the MPI version of the NAG Parallel Library, please read Section 1.3.4.1 and you may then skip Section 1.3.4.2.

If you intend to use the PVM version of the NAG Parallel Library, you may skip Section 1.3.4.1, but please read Section 1.3.4.2.

1.3.4.1 The NAG Parallel Library/MPI

The NAG Parallel Library/MPI uses the Basic Linear Algebra Communication Subprograms (BLACS) [6] for most of the communication within the Library and makes explicit calls to MPI where BLACS does not provide the necessary functionalities. The version of the BLACS used in the Library uses MPI as its communication kernel (MPI-BLACS). It is therefore necessary to know how to run an MPI program on the machine, or machines, that you are planning to use. MPI does not standardise the environment within which your parallel program will be running. There are two basic types of parallel environments: parallel computers and clusters of workstations. Naturally, a parallel computer (usually) provides an integrated, relatively easy way of running a parallel program. Clusters of workstations and other computers, on the other hand, usually have no standard way of running a parallel program and will require some additional setup. However, certain implementations of MPI provide facilities which hide these differences behind some kind of script. For example the MPICH implementation of MPI provides the 'mpirun' script for running an MPI program on a cluster of worksations, see [9]. See also the Users' Note for your particular implementation.

For safe communication the Library operates on the BLACS **context** which is an equivalent terminology to an MPI **communicator** (but note that a BLACS context is not an MPI communicator). In the BLACS, each logical processor grid is enclosed in a context which can be viewed as a unique identifier for that logical grid. A context can also be thought of as a self-contained message passing space. This means that processors within the grid associated with a particular context can safely communicate even if other, possibly overlapping, grids are also communicating. This context is an argument to most of the Library routines which, in particular, allows users to call a BLACS routine themselves if so wished. The users can also call MPI routines with the NAG Parallel Library/MPI provided they know how to translate between the BLACS context and the MPI Communicator. The BLACS provide facilities which allows such translation. Interested users are referred to the Tutorial for an example. See also [15].

Nevertheless, you need not necessarily be concerned about the use of the BLACS or MPI (other than running tasks using MPI on your machine(s)), since the definition of a logical processor grid and its

context is handled by the NAG Parallel Library routine **Z01AAFP, which must be called prior to the use of the Library** on all processors. On completion of calls to NAG Parallel Library routines the Library routine **Z01ABFP must be called** on all processors in order to undefine the logical grid and the context. Thus, a typical sequence of calls for straightforward use of the NAG Parallel Library might be of the form:

```
      PROGRAM USER
*
*     Declarations and initialisation
      :
*
*     Initialise the NAG Parallel Library
      CALL Z01AAFP( ICNTXT, MP, NP, IFAIL )
      :
      CALL D01GCFP( ... )
      :
      CALL F07ADFP( ... )
      :
*
*     Undefine the NAG Parallel Library context
      CALL Z01ABFP( ICNTXT, 'No', IFAIL )
      END
```

Note that the above program is started on all the processes initiated by MPI. See the routine documents for Z01AAFP and Z01ABFP for further information, and see the Tutorial for examples of the use of Z01AAFP, especially examples of how to call Z01AAFP more than once.

Exceptions to the above are that calls may be made to certain utility routines in Chapter Z01 prior to calling Z01AAFP. For example the function Z01ACFP, which returns the value .TRUE. when called on the root processor may be called prior to a call to Z01AAFP. The use of Z01ACFP is illustrated in the Example Programs for each Library routine and in the Tutorial.

Since the Library assumes an SPMD model, all Library routines which have a context argument (usually ICNTXT) **must** be called by all processors. The context should be treated in a similar fashion to a pointer in that its value must not be altered (and there is no need to manipulate its value).

1.3.4.2 The NAG Parallel Library/PVM

The NAG Parallel Library/PVM uses the Basic Linear Algebra Communication Subprograms (BLACS) [6] for most of the communication within the Library. PVM is used to support the Library and it is necessary to know how to start PVM on the machine, or machines, that you are planning to use. See the Tutorial for a brief introduction to starting PVM.

The current version of the BLACS defines a **context**, which is intended to be a unique identifier for the logical grid used by a call to a BLACS, and means that the grid can safely communicate even if other, possibly overlapping, grids are also communicating. This context is an argument to most of the Library routines which, in particular, allows users to call a BLACS routine themselves if so wished.

Nevertheless, you need not necessarily be concerned about the use of the BLACS or PVM (other than starting PVM), since task spawning and the definition of a logical processor grid and its context is handled by the NAG Parallel Library routine **Z01AAFP, which must be called prior to the use of the Library** on all processors. On completion of calls to NAG Parallel Library routines the Library routine **Z01ABFP must be called** on all processors in order to undefine the logical grid and the context. Thus, a typical sequence of calls for straightforward use of the NAG Parallel Library might be of the form:

```
      PROGRAM USER
*
*     Declarations and initialisation
      :
```

```
*
*     Initialise the NAG Parallel PVM Library
      CALL Z01AAFP( ICNTXT, MP, NP, IFAIL )
      :
      CALL D01GCFP( ... )
      :
      CALL F07ADFP( ... )
      :
*
*     Undefine the NAG Parallel PVM Library context
      CALL Z01ABFP( ICNTXT, 'No', IFAIL )
      END
```

Note that when Z01AAFP is called the program (USER in the above example) is started on all the spawned processes. See the routine documents for Z01AAFP and Z01ABFP for further information, and see the Tutorial for examples of the use of Z01AAFP, especially examples of how to call Z01AAFP more than once.

Exceptions to the above are that calls may be made to certain utility routines in Chapter Z01 prior to calling Z01AAFP. For example the function Z01ACFP, which returns the value .TRUE. when called on the root processor may be called prior to a call to Z01AAFP. The use of Z01ACFP is illustrated in the Example Programs for each Library routine and in the Tutorial.

Since the Library assumes an SPMD model, all Library routines which have a context argument (usually ICNTXT) **must** be called by all processors. The context should be treated in a similar fashion to a pointer in that its value must not be altered (and there is no need to examine its value).

1.4 Releases of the Library

Periodically there is a new **Release** of the NAG Parallel Library: new routines are added, corrections or improvements are made to existing routines; occasionally routines are withdrawn if they have been superseded by improved routines.

At each release, the documentation of the Library is updated. You must make sure that your documentation has been updated to the same release as the Library software that you are using.

Releases are numbered, e.g. 1, 2. The current release is 2.

The Library software may be updated between releases to an intermediate maintenance level, in order to incorporate corrections. Maintenance levels are indicated by a letter following the release number, e.g. 1A, 1B, and so on (Release 1 documentation supports all these maintenance levels).

1.5 Implementations of the Library

The NAG Parallel Library is available on a number of different computer systems. For each distinct system, an **implementation** of the Library is prepared by NAG, e.g. the IBM SP2 implementation. The implementation is distributed to sites as a tested compiled library.

An implementation is usually specific to a range of machines (e.g. the Sun SPARC range); it may also be specific to a particular operating system, Fortran compiler, compiler option or specific implementation of MPI or PVM (e.g. scalar or vector mode, or MPICH). Essentially the same facilities are provided in all implementations of the Library, but, because of differences in arithmetic behaviour and in the compilation system, routines cannot be expected to give identical results on different systems, especially for sensitive numerical problems.

The documentation supports all implementations of the Library, with the help of a few simple conventions, and a small amount of implementation-dependent information, which is published in a separate **Users' Note** for each implementation (see Section 3.4).

See Section 1.8 for further information about networks of machines.

The current MPI version of the Library requires the use of MPI-1 Standard. A public domain version of MPI (known as 'MPICH') is available from [16]. The current PVM version of the Library requires the use of PVM, Release 3.3 or later. A public domain version of PVM is available from netlib [5] and [3].

1.6 Precision of the Library

The NAG Parallel Library is developed principally as a **double precision** version and the Manual documents this version. But a **single precision** version may be available and you should consult the Users' Note to ascertain the precision of your implementation, and for information about the interpretation of precision dependent terms in the Manual.

1.7 Library Identification

You must know **which implementation** and **which release** of the Library you are using or intend to use. To find out which implementation and release of the Library is available at your site, you can run a program which calls the NAG Parallel Library routine A00AAFP. This routine has no arguments; it simply outputs text on the root processor to the NAG Parallel Library advisory message unit (see Section 2.4). An example of the output is:

```
** Start of NAG Parallel Library implementation details **

   Implementation title: Silicon Graphics IRIX 5
           Product Code: FDSG502M
                Release: 2
              Precision: FORTRAN DOUBLE PRECISION
Message passing library: MPICH implementation of MPI

** End of NAG Parallel Library implementation details **
```

The product code can be ignored, except possibly when communicating with NAG; see Section 5.

1.8 Implementations on Networks of Machines

Currently the NAG Parallel Library is only supported on homogeneous networks, but we aim to remove this restriction in the future and, at least, to support networks in which each machine uses IEEE arithmetic [2].

1.9 Fortran Language Standards

All the Library routines conform to ANSI Standard Fortran 77 [14], except for the use of names longer than 6 characters. In particular, as described in Section 1.1, most of the documented routines have 7-character names. Most of the BLACS, MPI and PVM routines, which are called by many of the Library routines, also have names longer than 6 characters.

1.10 Relationship to the NAG Fortran Library

The NAG Fortran Library is a comprehensive library of numerical and statistical routines that is available on a wide range of serial, vector and shared memory machines, including high performance workstations, and is available on a number of distributed memory machines as a single node implementation. You may wish to consider using the NAG Fortran Library on single nodes, in conjunction with the NAG Parallel Library. See the Tutorial for an example of calling the NAG Fortran Library.

Routines in the NAG Parallel Library that have the same functionality as a corresponding routine in the NAG Fortran Library have the same name, except for the addition of the last character P in the name of the routine in the NAG Parallel Library. A few routines in the NAG Parallel Library are identical to routines in the NAG Fortran Library, in which case their names are the same. These are principally routines in Chapters X01, X02 and X04 which return information such as mathematical constants, relative machine precision, and channel numbers for output.

1.11 Relationship Between the MPI and PVM Versions of the Library

In terms of functionality, the routines in both versions of the NAG Parallel Library are the same. However, the major difference between these two libraries is in the contrasting ways in which processes are started.

For PVM, processes are spawned dynamically during execution of the user's job, and the user may do this himself or call a Library routine. MPI, like most message passing libraries, assumes a static set of processors at load time.

The only difference in content between the two versions of the Library is that the MPI version contains the two routines Z01AEFP and Z01BGFP, whilst the equivalent routines in the PVM version are Z01ADFP and Z01BDFP.

2 Using the Library

2.1 General Advice

A NAG Parallel Library routine **cannot** be guaranteed to return meaningful results, irrespective of the data supplied to it. Care and thought **must** be exercised in:

(a) formulating the problem;

(b) programming the use of library routines, including the distribution of data;

(c) assessing the significance of the results.

Section 2.2 to Section 2.5 are concerned with programming the use of library routines.

2.2 Programming Advice

The NAG Parallel Library and its documentation are designed on the assumption that you know how to write a calling program in Fortran 77 and how to invoke MPI or PVM. Some examples are given in the Tutorial and an example of use is given in Section 8 of most routine documents.

When programming a call to a routine, read the routine document carefully, especially the description of the **Arguments**. This states clearly which arguments must have values assigned to them on entry to the routine, which return useful values on exit and whether the arguments are local or global. See Section 3.3 for further guidance.

The most common types of programming error in using the Library are:

– incorrect arguments in a call to a Library routine;
– calling a double precision implementation of the Library from a single precision program, or vice versa;
– incorrect data distribution.

Therefore if a call to a Library routine results in an unexpected error message from the system (or possibly from within the Library), **check** the following:

Has the NAG routine been called with the correct number of arguments?

Do the arguments all have the correct type?

Have all array arguments been dimensioned correctly?

Is your program in the same precision as the NAG Parallel Library routines to which your program is being linked?

Has the data been correctly distributed?

Avoid the use of NAG-type names for your own program units or COMMON blocks: in general, do not use names which contain a three-character NAG chapter name embedded in them; they may clash with the names of an auxiliary routine or COMMON block used by the NAG Parallel Library.

2.3 Error-handling and the Argument IFAIL

NAG Parallel Library routines may detect various kinds of error, failure or warning conditions. Such conditions are handled in a systematic way by the Library. They fall roughly into four classes:

(i) the utility routine Z01AAFP has not been called (see Section 1.3.4) or an invalid context has been supplied;

(ii) an invalid value of an argument on entry to a routine, or different input values for a global argument;

(iii) a numerical failure during computation (e.g. approximate singularity of a matrix, failure of an iteration to converge);

(iv) a warning, that although the computation has been completed, the results cannot be guaranteed to be completely reliable.

All four classes are handled in the same way by the Library, and are all referred to here simply as 'errors'.

The error-handling mechanism uses the argument IFAIL, which occurs in the calling sequence of most NAG Parallel Library routines (almost always it is the last argument). IFAIL serves two purposes:

(a) it allows users to specify what action a Library routine should take if it detects an error;

(b) it reports the outcome of a call to a Library routine, either success (IFAIL = 0) or failure (IFAIL $\neq$ 0, with different values indicating different reasons for the failure, as explained in Section 5 of the routine document).

For the first purpose IFAIL **must** be assigned a value before calling the routine; since IFAIL is reset by the routine, it **must** be passed as a variable, not as an integer constant. Allowed values on entry are:

IFAIL = 0: an error message is output, and execution is terminated ('hard failure');

IFAIL = + 1: execution continues without any error message;

IFAIL = − 1: an error message is output, and execution continues.

The settings IFAIL = $\pm$ 1 are referred to as 'soft failure'.

The safest choice when only one grid is present is to set IFAIL to 0, but this is not always convenient: some routines return useful results even though a failure (in some cases merely a warning) is indicated. The specification of IFAIL in Section 4 of a routine document suggests a suitable setting of IFAIL for that routine. However, if IFAIL is set to $\pm$ 1 on entry, it is **essential** for the program to test its value on exit from the routine, and to take appropriate action. See Section 2.4 below for information on the output of error messages.

In the case of hard failure, all BLACS processes will stop, including all those which are in the Library context. (Users familiar with the BLACS might like to note that the routine BLACS_ABORT is called first.)

See the Tutorial for examples of error handling.

Routines in Chapters F07 and F08 do **not** use the usual error handling mechanism; in order to preserve complete compatibility with ScaLAPACK software, they have a diagnostic output argument INFO which need not be set before entry. The ScaLAPACK routines perform a soft failure, so it is essential to test INFO on exit from a ScaLAPACK routine. But, an error message is output when INFO $\neq$ 0 on return. See the F07 or F08 Chapter Introductions for further details.

2.4 Input/Output in the Library

Most NAG Parallel Library routines perform no output to an external file, except possibly to output an error message. All error messages are written, only by the root process, to a logical **error message** unit. This unit number (which is set by default to 6 in most implementations) can be changed by calling the Library routine X04AAF.

Some Library routines (e.g.A00AAFP) output messages, only by the root process, to a logical **advisory message** unit. This unit number (which is also set by default to 6 in most implementations) can be changed by calling the Library routine X04ABF. Although it is logically distinct from the error message unit, in practice the two unit numbers may be the same.

All output from the Library is formatted.

You must ensure that the relevant Fortran unit number is associated with the desired external file, either by an OPEN statement in your calling program, or by operating system commands.

If more than one grid is defined, then you **must** ensure that one of the processors in the Library Grid is attached to the terminal, otherwise error messages may be lost. (See routine Z01AAFP for further information.)

2.5 Auxiliary Routines

In addition to those Library routines which are documented and are intended to be called by users, the Library also contains many auxiliary routines.

In general, you need not be concerned with them at all, although you may be made aware of their existence if, for example, you examine a memory map of an executable program which calls NAG routines.

NAG auxiliary routines have names which are similar to the name of the documented routine(s) to which they are related, but with the sixth letter 'Z', 'Y', and so on, e.g.

D01AUTP is an auxiliary routine called by D01AUFP.

The Library also calls a number of **de facto** standard routines such as the BLACS, PBLAS (see [4]) and BLAS (see for example [1] and the references contained there). A few MPI or PVM routines are also called. MPI routines that are called by the Library have names that start with MPI; to avoid unintentional clashes with MPI names you should not declare names that start with these characters. PVM routines that are called by the Library have names that start with PVM or LPVM; to avoid unintentional clashes with PVM names you should not declare names that start with these characters. Similarly, a number of the BLACS routines start with the characters BLACS. Additional names to avoid, including further BLACS names, are listed in the Reserved Names document of the Manual.

See the Users' Note for details of whether or not external libraries, such as the BLACS or MPI or PVM, need to be linked with the NAG Parallel Library.

3 Using the Documentation

3.1 Using the Manual

The Manual is designed to serve the following functions:

- to give background information about different areas of numerical and statistical computation;
- to advise on the choice of the most suitable NAG Parallel Library routine or routines to solve a particular problem;
- to give all the information needed to call a NAG Parallel Library routine correctly from a Fortran program, and to assess the results.

At the beginning of the Manual are some general introductory documents, namely this Essential Introduction and the Tutorial. The following documents may help you to find the chapter, and possibly the routine, which you need to solve your problem:

Contents Summary	–	a structured list of routines in the Library, by chapter;
KWIC Index	–	a keyword index to chapters and routines;
GAMS Index	–	a list of Library routines classified according to the GAMS scheme.

Having found a likely chapter or routine, you should read the corresponding **Chapter Introduction,** which gives background information about that area of numerical computation, and recommendations on the choice of a routine, including indexes or tables.

When you have chosen a routine, you must consult the **routine document**. Each routine document is essentially self-contained (it may contain references to related documents). It generally includes a description of the method, detailed specifications of each argument, explanations of each error exit, remarks on accuracy, and an example program to illustrate the use of the routine.

3.2 Structure of Routine Documents

All routine documents have the same structure, consisting of eight numbered sections:

1. Description
2. Specification
3. Data Distribution
4. Arguments (see Section 3.3 below)
5. Errors and Warnings

6. Further Comments
7. References
8. Example (see Section 3.5 below)

3.3 Specifications of Arguments

Section 4 of each routine document contains the specification of the arguments, in the order of their appearance in the argument list.

3.3.1 Classification of arguments

Arguments are classified as follows:

Input: you must assign values to these arguments on or before entry to the routine, and these values are unchanged on exit from the routine.

Output: you need not assign values to these arguments on or before entry to the routine; the routine may assign values to them.

Input/Output: you must assign values to these arguments before entry to the routine, and the routine may then change these values.

Workspace: array arguments which are used as workspace by the routine. You must supply arrays of the correct type and dimension. In general, you need not be concerned with their contents.

External Procedure: a subroutine or function which must be supplied (e.g. to evaluate an integrand or to print intermediate output). Usually it must be supplied as part of your calling program, in which case its specification includes full details of its argument list and specifications of its arguments (all enclosed in a box). Its arguments are classified in the same way as those of the Library routine, but because you must write the procedure rather than call it, the significance of the classification is different:

Input: values may be supplied on entry, which your procedure **must not** change.

Output: you should assign values to these arguments before exit from your procedure.

Input/Output: values may be supplied on entry, and you should assign values to them before exit from your procedure.

In addition arguments are also classified as either *Local* or *Global*. See Section 1.3.3 for a definition of these terms. Thus a global input argument will be classified as *Global Input*. All external procedures are global (i.e., the same procedure must be supplied on each processor).

3.3.2 Constraints and suggested values

The word '*Constraint*:' or '*Constraints*:' in the specification of an *Input* argument introduces a statement of the range of valid values for that argument, e.g.

Constraint: $N > 0$.

If the routine is called with an invalid value for the argument (e.g. $N = 0$), the routine will take an error exit, returning a non-zero value of IFAIL (see Section 2.3).

Constraints on arguments of type CHARACTER only list uppercase alphabetic characters, e.g.

Constraint: STRING = 'A' or 'B'.

In practice all routines with CHARACTER arguments will permit the use of lower-case characters.

The phrase '*Suggested Value*:' introduces a suggestion for a reasonable initial setting for an *Input* argument (e.g. accuracy or maximum number of iterations) in case you are unsure what value to use; you should be prepared to use a different setting if the suggested value turns out to be unsuitable for your problem.

3.3.3 Array Arguments

Most array arguments have dimensions which depend on the size of the problem. In Fortran terminology they have 'adjustable dimensions': the dimensions occurring in their declarations are integer variables which are also arguments of the Library routine.

For example, a Library routine might have the specification:

```
SUBROUTINE [NAME](M, N, A, B, LDB)
INTEGER       M, N, A(N), B(LDB,N), LDB
```

For a **1-d** array argument, such as A in this example, the specification would begin:

A(N) − INTEGER array.

You must ensure that the dimension of the array, as declared in your calling (sub)program, is at least as large as the value you supply for N. It may be larger; but the routine uses only the first N elements.

For a **two-dimensional** array argument, such as B in the example, the specification might be:

B(LDB,N) − INTEGER array.

On entry: the m by n matrix B.

and the argument LDB might be described as follows:

LDB − INTEGER. *Input*

On entry: the first dimension of the array B as declared in the (sub)program from which `[NAME]` is called.

Constraint: LDB $\geq$ M.

You **must** supply the **first** dimension of the array B, as declared in your calling (sub)program, through the argument LDB, even though the number of rows actually used by the routine is determined by the argument M. You must ensure that the first dimension of the array is at least as large as the value you supply for M. The extra argument LDB is needed because Fortran does not allow information about the dimensions of array arguments to be passed automatically to a routine.

You must also ensure that the **second** dimension of the array, as declared in your calling (sub)program, is at least as large as the value you supply for N. It may be larger, but the routine only uses the first N columns.

A program to call the hypothetical routine used as an example in this section might include the statements:

```
INTEGER AA(100), BB(100,50)
LDB = 100
.
.
.
M = 80
N = 20
CALL  [NAME](M,N,AA,BB,LDB)
```

Fortran requires that the dimensions which occur in array declarations must be greater than zero. Many NAG routines are designed so that they can be called with an argument like N in the above example set to 0 (in which case they would usually exit immediately without doing anything). If so, the declarations in the Library routine would use the 'assumed size' array dimension, and would be given as:

```
INTEGER        M, N, A(*), B(LDB,*), LDB
```

However, the original declaration of an array in your calling program must always have constant dimensions, greater than or equal to 1.

Local arrays may have different dimensions on different logical processors (but must all meet the constraints on minimum size).

Consult an expert or a textbook on Fortran if you have difficulty in calling Library routines with array arguments.

3.4 Implementation-dependent Information

In order to support all implementations of the Library, the Manual has adopted a convention of using ***bold italics*** to distinguish terms which may have different interpretations in different implementations.

An important bold-italicised term is ***machine precision***, which denotes the relative precision to which floating-point numbers are stored in the computer; e.g. in an implementation with approximately 16 decimal digits of precision, ***machine precision*** has a value of approximately 10^{-16}.

The precise value of ***machine precision*** is given by the function X02AJF. Other functions in Chapter X02 return the values of other implementation-dependent constants, such as the overflow threshold, or the largest representable integer. Refer to the X02 Chapter Introduction for more details.

For each implementation of the Library, a separate **Users' Note** is published. This is a short document, revised at each release. At most installations it is available in machine-readable form. It gives any necessary additional information which applies specifically to that implementation, in particular:

- the interpretation of bold italicised terms;
- the values returned by X02 routines;
- the default unit numbers for output (see Section 2.4);
- details of name changes for Library routines (see Section 1.7 and Section 2.5).

3.5 Example Programs and Results

The **Example Program** in Section 8 of each routine document illustrates a simple call of the routine. The programs are designed so that they can fairly easily be modified, and so serve as the basis for a simple program to solve a user's own problem.

For each implementation of the Library, NAG distributes the example programs in machine-readable form, with all necessary modifications already applied. Many sites make the programs accessible in this form to users.

Note that the results from running the example programs may not be identical in all implementations, and may not agree exactly with the results which are printed in the Manual and which were obtained from a Silicon Graphics double precision implementation (with approximately 16 digits of precision).

The Users' Note for your implementation will mention any special changes which need to be made to the example programs, and any significant differences in the results.

4 Summary for New Users

If you are unfamiliar with the NAG Parallel Library and are thinking of using a routine from it, please follow these instructions:

(a) read the **Tutorial**;

(b) read the whole of the **Essential Introduction**;

(c) consult the **Contents Summary**, **KWIC Index** or **GAMS Index** to choose an appropriate chapter or routine

(d) read the relevant **Chapter Introduction**;

(e) choose a routine, and read the **routine document**. If the routine does not after all meet your needs, return to steps (c) or (d);

(f) read the **Users' Note** for your implementation;

(g) consult local documentation, which should be provided by your local support staff, about access to the NAG Parallel Library and other associated libraries, such as the BLACS, on your computing system.

You should now be in a position to include a call to the routine in a program, and to attempt to run it. You may of course need to refer back to the relevant documentation in the case of difficulties, for advice on assessment of results, and so on.

As you become familiar with the Library, some of steps (a) to (g) can be omitted, but it is always essential to:

– be familiar with the Chapter Introduction;
– read the routine document;
– be aware of the Users' Note for your implementation.

5 Contact between Users and NAG

For further advice or communication about the NAG Parallel Library, you should first turn to the staff of your local computer installation. This covers such matters as:

– obtaining a copy of the Users' Note for your implementation;
– obtaining information about local access to the Library;
– seeking advice about using the Library;
– reporting suspected errors in routines or documents;
– making suggestions for new routines or features;
– purchasing NAG documentation.

Your installation may have advisory and/or information services to handle such enquiries. In addition NAG asks each installation mounting the Library to nominate a NAG **site representative**, who may be approached directly in the absence of an advisory service. Site representatives receive information from NAG about confirmed errors, the imminence of updates, and so on, and will forward users' enquiries to the appropriate person in the NAG organisation if they cannot be dealt with locally. If you are unable to make contact with your local site representative please contact NAG directly. Full details of the NAG Response Centres are given in the Users' Note and at the front of this Manual.

6 References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] ANSI/IEEE (1985) IEEE standard for binary floating-point arithmetic *Std 754-1985* IEEE, New York

[3] Browne S, Dongarra J J, Grosse E, Green S, Moore K, Rowan T and Wade R (1994) Netlib services and resources *Technical Report CS-94-222* Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA

[4] Choi J, Dongarra J J, Ostrouchov S, Petitet A P, Walker D W and Whaley R C (1994) The Design and Implementation of the ScaLAPACK LU, QR and Cholesky Factorization Routines *LAPACK Working Note 80. Technical Report CS-94-246* Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA

[5] Dongarra J J and Grosse E (1987) Distribution of mathematical software via electronic mail *Comm. ACM* **30** 403–407

[6] Dongarra J J and Whaley R C (1995) A users' guide to the BLACS v1.0. *LAPACK Working Note 94 (Technical Report CS-95-281)* Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA.
URL: http://www.netlib.org/lapack/lawns/lawn94.ps

[7] Geist A, Beguelin A, Dongarra J J, Jiang W, Manchek R and Sunderam V (1994) *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing* The MIT Press, Cambridge, MA, USA

[8] Gropp W, Lusk E and Skjellum A (1994) *Using MPI: Portable Parallel Programming with the Message-Passing Interface* MIT Press, Cambridge, MA

[9] Gropp W nad Lusk E (1994) *Users' Guide for mpich, a Portable Implementation of MPI.* Mathematics and Computer Science Division, Argonne National Lab., Illinois, USA.

[10] Koelbel C H, Loveman D B, Schreiber R S, Steele Jr. G L, and Zosel M E (1994) *The High Performance Fortran Handbook* The MIT Press, Cambridge, MA, USA

[11] McBryan O A (1994) An overview of message passing environments *Parallel Comput.* **20** 417–444

[12] Message Passing Interface Forum (1994) *MPI: A Message-Passing Interface Standard* Technical Report CS-94-230, University of Tennessee, Knoxville, TN

[13] (1960–1976) Collected algorithms from ACM index by subject to algorithms

[14] (1978) American National Standard Fortran *Publication X3.9* American National Standards Institute

[15] Whaley R C (1995) *Using BLACS and MPI in ScaLAPACK.*, Univ. of TN, Knoxville, TN 37996.

[16] Web address: *http://www.mcs.anl.gov/mpi/*