NAG Fortran Library

Thread Safety

1 Multithreaded Applications and Thread Safety

A thread is a basic entity to which an operating system allocates CPU time. A thread has its own registers, stack and process resources. Threads provide a convenient way of allowing an application to maximise its usage of CPU resources in a system, especially in a multiple processor configuration. A routine is termed 'thread safe' if it can be called safely from two or more concurrently running threads.

The remainder of this document describes thread safety within the context of the NAG Fortran Library and provides guidelines for calling Library routines from multithreaded applications.

2 Thread Safety and the NAG Fortran Library

It is essential that you refer to the Users' Note for details of whether the Library has been compiled in a manner that facilitates the use of multiple threads. Also, your local site may have decided only to install a Library of thread safe routines; please contact your site installer for details of the installation.

2.1 Thread Safe Constructs

In a Fortran 77 context the constructs that prohibit thread safety are, potentially, DATA, SAVE, COMMON and EQUIVALENCE. This is because such constructs define data that may be shared by different threads, perhaps leading to unwanted interactions between them: for example, the possibility that one thread may be modifying the contents of a COMMON block at the same time as another thread is reading it. You are therefore advised to use such constructs with great care and to avoid their use wherever possible within multithreaded applications.

At Mark 20 of the NAG Fortran Library the thread safe provision has been significantly enhanced by

(a) eliminating unsafe constructs wherever possible to make the majority of routines safe for use in multithreaded applications;

(b) providing equivalent thread safe routines with the same functionality where complete removal of unsafe constructs would affect interface design. Two approaches have been taken to provide thread safe equivalents; see Section 2.2 for further details.

See Section 3.2 for a list of the remaining routines that are currently thread unsafe with no thread safe equivalent. It should be noted that it is always safe to call the NAG Library in one thread (only) of a multithreaded application.

2.2 Library Routines with Thread Safe Equivalents

At Mark 20 of the NAG Fortran Library two approaches have been taken to provide thread safe equivalents to routines containing unsafe constructs. In the first approach a close connection between the original routine and the thread safe equivalent can be maintained, allowing the two routines to appear as a pair and share the same root name. In the second approach more fundamental changes in interface design have been made such that the correspondence between a routine and its thread safe equivalent cannot be maintained through the root name.

2.2.1 Routine and thread safe equivalent sharing the same root name

At Mark 20 of the NAG Fortran Library there are pairs of routines which share the same root name, for example, the routines E04UCF and E04UCA. Each routine in the pair has exactly the same functionality, except that one of them has additional parameters in order to make it safe for use in multithreaded applications. The routine that is safe for use in multithreaded applications has a different last character in the name in place of the usual character (typically 'A' instead of 'F'). Such pairs are documented via one routine document. If the pair of routines contain a routine argument in their interface then the routine with additional parameters will have parameter arrays that enable you to pass information to the routine

argument without the need for COMMON blocks. In some cases the routine with additional parameters may need to be initialised by a separate initialisation routine; this requirement will be clearly documented.

2.2.2 Other routines with thread safe equivalents

You will note that some of the equivalent routines listed in Section 3.1 do not share the same root name as the original routine containing unsafe constructs. In these cases you are advised to consult the relevant chapter introduction and routine documents for further information. You are further advised to consult the relevant entry in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

2.3 Routines with Routine Arguments

Some Library routines require you to supply a routine and to pass the name of the routine as an argument in the call to the Library routine. For many of these Library routines, the supplied routine interface includes array arguments specifically for you to pass information to the supplied routine. However, there remain some Library routines for which you may need to supply your provided routine with more information than can be given via the interface argument list. In such circumstances it is usual to define a COMMON block containing the required data in the supplied routine (and also in the calling program). It is safe to do this only if no data referenced in the defined COMMON block is updated within the supplied routine (thus avoiding the possibility of simultaneous modification by different threads). Where separate calls are made to a Library routine by different threads and these calls require different data sets to be passed through COMMON blocks to user-supplied routines, these routines and the COMMON blocks defined within them should have different names.

You are advised to check, in the relevant chapter introduction, whether the Library routines you intend to call have equivalent reverse communication interfaces. These have been designed specifically for problems where user-supplied routine interfaces are not flexible enough for a given problem, and their use should eliminate the need to provide data through COMMON blocks.

2.4 Input/Output

The Library contains routines for setting the current error and advisory message unit numbers (X04AAF and X04ABF). These routines use the SAVE statement to retain the values of the current unit numbers between calls. It is therefore not advisable for different threads of a multithreaded program to set the message unit numbers to different values. A consequence of this is that error or advisory messages output simultaneously may become garbled, and in any event there is no indication of which thread produces which message. You are therefore advised always to select the 'soft failure' mechanism without any error message (IFAIL $= +1$, see Section 2.3 of the Essential Introduction) on entry to each NAG routine called from a multithreaded application; it is then essential that the value of IFAIL be tested on return to the application.

A related problem is that of multiple threads writing to or reading from files. You are advised to make different threads use different unit numbers for opening files and to give these files different names (perhaps by appending an index number to the file basename). The only alternative to this is for you to protect each write to a file or unit number; for example, by putting each WRITE statement in a critical region.

2.5 Implementation Issues

In some implementations of the NAG Library calls are made to vendor BLAS and/or LAPACK Library routines. Although NAG perform tests to ensure that these calls are behaving correctly on multiple threads, NAG cannot guarantee the thread safety of the vendor BLAS and LAPACK routines. You are advised to refer to the Users' Note for details of whether the Library is to be linked with vendor BLAS and/or LAPACK Libraries.

3 Lists of Thread Unsafe Routines

3.1 Thread Unsafe Routines with Thread Safe Equivalents

At Mark 20 the routines listed in the following table are not thread safe in any implementations, but do have equivalents that are safe to use in multithreaded applications (also listed).

Routine	Thread Safe Equivalent	Routine	Thread Safe Equivalent	Routine	Thread Safe Equivalent
C05PDF	C05PDA	D03PCF	D03PCA	D03PDF	D03PDA
D03PHF	D03PHA	D03PJF	D03PJA	D03PPF	D03PPA
E04ABF	E04ABA	E04BBF	E04BBA	E04CCF	E04CCA
E04DGF	E04DGA	E04DJF	E04DJA	E04DKF	E04DKA
E04MFF	E04MFA	E04MGF	E04MGA	E04MHF	E04MHA
E04NCF	E04NCA	E04NDF	E04NDA	E04NEF	E04NEA
E04NFF	E04NFA	E04NGF	E04NGA	E04NHF	E04NHA
E04NKF	E04NKA	E04NLF	E04NLA	E04NMF	E04NMA
E04UCF	E04UCA	E04UDF	E04UDA	E04UEF	E04UEA
E04UFF	E04UFA	E04UGF	E04UGA	E04UHF	E04UHA
E04UJF	E04UJA	E04UNF	E04USA	E04UQF	E04UQA
E04URF	E04URA	E04USF	E04USA	E04XAF	E04XAA
E04ZCF	E04ZCA	F11BAF	F11BDF	F11BBF	F11BEF
F11BCF	F11BFF	F11GAF	F11GDF	F11GBF	F11GEF
F11GCF	F11GFF	G05CAF	G05KAF	G05CBF	G05KBF
G05CCF	G05KCF	G05CFF	not required	G05CGF	not required
G05DAF	G05LGF	G05DBF	G05LJF	G05DCF	G05LNF
G05DDF	G05LAF	G05DEF	G05LKF	G05DFF	G05LLF
G05DHF	G05LCF	G05DJF	G05LBF	G05DKF	G05LDF
G05DPF	G05LMF	G05DRF	G05MKF	G05DYF	G05MAF
G05DZF	G05KEF	G05EGF	G05PAF	G05EHF	G05NAF
G05EJF	G05NBF	G05EWF	G05PAF	G05EXF	G05MZF
G05EYF	G05MZF	G05EZF	G05LZF	G05FAF	G05LGF
G05FBF	G05LJF	G05FDF	G05LAF	G05FEF	G05LEF
G05FFF	G05LFF	G05FSF	G05LPF	G05GAF	G05QAF
G05GBF	G05QBF	G05HDF	G05PCF	G05ZAF	not required

3.2 Thread Unsafe Routines with No Thread Safe Equivalents

At Mark 20 the routines listed in the following table are **not** thread safe in any implementations and **do not** as yet have thread safe equivalents.

C05NDF	D01GBF	D01GCF	D01GDF	D02BGF	D02BHF
D02BJF	D02CJF	D02EJF	D02GAF	D02GBF	D02HAF
D02HBF	D02JAF	D02JBF	D02KAF	D02KDF	D02KEF
D02LAF	D02LXF	D02LYF	D02LZF	D02MZF	D02NBF
D02NCF	D02NDF	D02NGF	D02NHF	D02NJF	D02NMF
D02NNF	D02NSF	D02NTF	D02NUF	D02PCF	D02PDF
D02PVF	D02PWF	D02PXF	D02PYF	D02PZF	D02QFF
D02QGF	D02QWF	D02QXF	D02QYF	D02QZF	D02RAF
D02SAF	D02XJF	D02XKF	D03PEF	D03PFF	D03PKF
D03PLF	D03PRF	D03PSF	D03PUF	D03PVF	D03PWF
D03PXF	D03RAF	D03RBF	D05BDF	D05BEF	E01SBF
F04YCF	F04ZCF	G01DHF	G01EMF	G01FMF	G01HBF
G01JDF	G03FCF	G04DBF	G08EAF	G08EBF	G08ECF
G08EDF	G10BAF	G13DCF	H02BBF	H02BFF	H02BVF
H02CBF	H02CCF	H02CDF	H02CEF	H02CFF	H02CGF
X04AAF	X04ABF				