NAG 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 maximize 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 from two or more concurrently running threads without compromising results.

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

2 Thread Safety and the NAG 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.

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 Library the thread safe provision was 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.

At subsequent Marks of the NAG Library, the list of potentially thread unsafe routines will be reduced as these routines are replaced by thread safe equivalents. Within a small number of Marks there will be no potentially thread unsafe routines in the Library.

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 any NAG Library routine in one thread (only) of a multithreaded application.

2.2 Library Routines with Thread Safe Equivalents

At Marks 21 and 22 of the NAG 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 22 of the NAG Library there are pairs of routines which share the same root name, for example, the routines E04UCF/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 user-supplied routine argument in their interface then the 'A' routine will have array arguments that enable you to pass information to the user-supplied routine without the need for `COMMON` blocks. In some cases the 'A' routine may need to be initialized by a separate initialization routine; this requirement will be clearly documented.

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 actual argument in the call to the Library routine. For many of these Library routines, the supplied routine interface includes array parameters (called IUSER and RUSER) 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.

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. Where reverse communication interfaces are not available, it is usual to define a `COMMON` block containing the required data in the supplied routine (and also in the calling program). It is threadsafe to do this only if no data referenced in the defined `COMMON` block is updated (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 must have different names.

Thread safety of user-supplied routines is also an issue with a number of routines in multi-threaded implementations of the NAG Library, which may internally parallelize around the calls to the user-supplied routines, affecting not just COMMON blocks but also how the IUSER and RUSER arrays may be used. In these cases, use of thread-unsafe features within the user-supplied routines should be avoided wherever possible, and care should be taken to ensure any usage of IUSER and RUSER is undertaken in a thread-safe manner. If they cannot be avoided entirely, you are advised to contact NAG for assistance on how to manage possible data dependencies within shared data structures.

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 ($\mathrm{IFAIL} = +1$, see Section 3.3 in the Essential Introduction) on entry to each NAG Library 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 from your multithreaded application. 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 25 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).

Unsafe Routine	Thread Safe Equivalent
D02PCF	D02PEF
D02PDF	D02PFF
D02PVF	D02PQF
D02PWF	D02PRF
D02PXF	D02PSF
D02PYF	D02PTF
D02PZF	D02PUF
D03PCF	D03PCA
D03PDF	D03PDA
D03PHF	D03PHA
D03PJF	D03PJA
D03PPF	D03PPA
E04ABF	E04ABA
E04BBF	E04BBA
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
E04UQF	E04UQA
E04URF	E04URA
E04USF	E04USA
E04XAF	E04XAA
F04YCF	F04YDF
F04ZCF	F04ZDF

3.2 Thread Unsafe Routines with No Thread Safe Equivalents

At Mark 25 the following routines listed are **not** thread safe in any implementations. If this is a difficulty then please contact NAG.

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
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
G08EAF
G08EBF
G08ECF
G08EDF
G10BAF
H02BBF
H02BFF
H02BVF
H02CBF
H02CCF
H02CDF
H02CEF
H02CFF
H02CGF
X04AAF
X04ABF
