NAG Library

Introduction to the NAG Library for SMP & Multicore

1 What is the NAG Library for SMP & Multicore?

The NAG Library for SMP & Multicore is a library of numerical routines intended for use on Symmetric Multiprocessor (SMP) machines, which are characterised by having both:

– a number of homogeneous processors (which may also be referred to as cores);

– a cache-coherent (real or virtual) shared memory accessible by all the processors (or cores).

Most current processors are multicore, i.e., they include more than one core on each chip. The vast majority of these have the necessary characteristics to be programmed with SMP techniques, and thus would be suitable for use with the NAG Library for SMP & Multicore. A small number of more specialised multicore processors cannot be used in this manner, and thus are **not** suitable for use with the NAG Library for SMP & Multicore. If in doubt, please contact NAG for advice on suitability.

The NAG Library for SMP & Multicore contains the full functionality currently available in the NAG Fortran Library, and users are encouraged to familiarise themselves with the Essential Introduction for a general overview of the structure of these products. Routine interfaces are mostly identical to those of the NAG Fortran Library (see Section 2.2 for details of the few differences that do exist). This makes the migration from using the NAG Fortran Library to using the NAG Library for SMP & Multicore trivial.

Many routines have been specially tuned for this Library to make use of the processing power and shared memory parallelism of SMP systems. Many other routines in the NAG Library for SMP & Multicore benefit from this increased performance by calling one or more of the tuned routines.

The list of routines that may benefit from SMP parallelism is listed in the 'Tuned and Enhanced Routines in the NAG Library for SMP & Multicore' document, and includes many key routines in the areas of:

Dense and Sparse Linear Algebra

FFTs

Random Number Generators

Quadrature

Partial Differential Equations

Interpolation

Curve and Surface Fitting

Correlation and Regression Analysis

Multivariate Methods

Time Series Analysis

Financial Option Pricing

Global Optimization

Wavelet Transforms

Matrix Functions

At each new Mark of the Library, we seek to expand the scope of parallelism to as many additional routines as possible, as well as incorporating new functionality introduced in the equivalent Mark of the NAG Fortran Library. Details of changes to the Library in the current Mark are available in the 'Mark 24 NAG Library for SMP & Multicore News' document.

1.1 Parallel Technology used within the NAG Library for SMP & Multicore

Parallelism within the NAG Library for SMP & Multicore is implemented using OpenMP, a portable API for shared memory programming that is available in many different compilers on a wide range of different hardware platforms. In general, users do not need to be aware of the specifics of how this is used within the library, beyond the additional steps needed to link, execute and get best performance from your code documented in Sections 2.1 and 2.2 below. However, users who are interested in the details of OpenMP should consult the documentation on the OpenMP website for further information.

Users who are calling NAG routines from within another threading mechanism need to be aware of whether or not this threading mechanism is compatible with OpenMP on their platform(s) of choice. The Users' Note document for each of the implementations in question will include some guidance on this, and users should contact NAG for further advice if required.

2 How to Use the NAG Library for SMP & Multicore

2.1 Linking and Executing Your Code

If your code currently contains calls to NAG Fortran Library routines then for most parallelized routines it is a simple matter of relinking your code to the NAG Library for SMP & Multicore (in place of the NAG Fortran Library) to benefit from the optimized performance. Exceptions to this general rule, where code changes may be necessary, are noted in Section 2.2 below.

Parallelism is requested by setting the appropriate environment variable (usually OMP_NUM_THREADS) to the desired number of threads you wish the routines to run on and then running your linked code. Generally the number of threads requested should not be more than the number of available (idle) cores on your system.

The steps required when compiling, linking and running programs on SMP machines, in order to fully exploit your parallelism are very much implementation specific. The particular details for your implementation are given in the Users' Note which should be read carefully before using the NAG Library for SMP & Multicore.

More general information regarding the conventions used in this Library is provided in the Essential Introduction.

2.2 How to Maximize the Performance of Your Application

There are a number of things you should consider when trying to maximize the performance of your code when linking to this Library. In the first instance you should be aware of the functionality of the Library and of which routines you should expect to achieve good levels of performance and scalability; for this you should consult the Tuned and Enhanced Routines in the NAG Library for SMP & Multicore document. There may be sections of your code which reproduce the functionality of a tuned/enhanced NAG routine or vendor BLAS routine; in such cases you should replace your sections of code with calls to the appropriate routines.

Note that the performance increase achieved, if any, when calling one of the tuned or enhanced routines will vary depending upon which routine is called, problem sizes and other parameters, system design and operating system configuration. If you frequently call a routine with similar data sizes and other parameters, it may be worthwhile to experiment with different numbers of threads, to determine the choice that gives optimal performance. Please contact NAG for further advice if required.

In addition there are areas of the NAG Library for SMP & Multicore that require further guidance, please see the following sections.

2.2.1 FFTs (Chapter C06)

In many implementations the vendors supply their own FFT routines that are optimized for their particular platforms. Where possible the NAG FFT routines call these vendor routines for optimal performance. Note that on some platforms additional workspace may be required compared to that listed in the Chapter C06 routine document. For details see the Users' Note for your implementation.

2.2.2 Quadrature (Chapter D01)

The performance of the quadrature routines in Chapter D01 depends upon the nature of the user-supplied function that calculates the value of the integrand at a given point and other problem parameters such as the relative accuracy required. Parallelism may not be beneficial for all problems, in particular the parallelism in D01GAF is only suitable for problems with a large number of data points.

2.2.3 PDEs (Chapter D03)

D03RAF and D03RBF require a user-supplied routine PDEDEF to evaluate the functions F_j, for $j = 1, 2, \ldots,$ NPDE. The parallelism within D03RAF and D03RBF will be more efficient if PDEDEF can also be parallelized. This is often the case, but you must add some OpenMP directives to your version of PDEDEF to implement the parallelism. For example, the body of code from the first test case in the document for D03RAF is

```
res(1:npts,1:npde) = ut(1:npts,1:npde) - diffusion*(uxx(1:npts,1:         &
  npde)+uyy(1:npts,1:npde)) - damkohler*(one+heat_release-u(1:npts,       &
  1:npde))*exp(-activ_energy/u(1:npts,1:npde))
```

This example can be parallelized, as the updating of RES in each iteration of the loop `I` over $1, \ldots,$ NPTS is independent of every other iteration. Thus this should be parallelized in OpenMP as follows

```
!$OMP DO
   Do i = 1, npts
      res(i,1:npde) = ut(i,1:npde) -diffusion*(uxx(i,1:npde)+uyy(i,1:npde &
       )) - damkohler*(1.0E0_nag_wp+heat_release-u(i,1:npde))*exp(-      &
        activ_energy/u(i,1:npde))
   End Do
!$OMP END DO
```

Note that the OpenMP PARALLEL directive must **not** be specified, as the OpenMP DO directive will bind to the PARALLEL region within the D03RAF or D03RBF code. Also note that this assumes the default OpenMP behaviour that all variables are SHARED, except for loop indices that are PRIVATE.

To avoid problems for existing library users, who will not have specified any OpenMP directives in their PDEDEF routine, the default assumption of D03RAF and D03RBF is that PDEDEF has not been parallelized, and they execute calls to PDEDEF in serial mode. You must indicate this fact by using the argument IND to D03RAF and D03RBF by adding 10 to the normal value. Thus, in the NAG Library for SMP & Multicore only, the following values may be specified for IND:

IND $= 0$
: Starts the integration in time. PDEDEF is assumed to be serial.

IND $= 1$
: Continues the integration after an earlier exit from the routine. In this case, only the following parameters may be reset between calls to D03RAF or D03RBF: TOUT, DT, TOLS, TOLT, OPTI, OPTR, ITRACE and IFAIL. PDEDEF is assumed to be serial.

IND $= 10$
: Starts the integration in time. PDEDEF is assumed to have been parallelized by you, as described above. In all other respects, this is equivalent to IND $= 0$.

IND $= 11$
: Continues the integration after an earlier exit from the routine. In this case, only the following parameters may be reset between calls to D03RAF or D03RBF: TOUT, DT, TOLS, TOLT, OPTI, OPTR, ITRACE and IFAIL. PDEDEF is assumed to have been parallelized by you, as described above. In all other respects, this is equivalent to IND $= 1$.

Constraint: $0 \leq$ IND ≤ 1 or $10 \leq$ IND ≤ 11.

On exit: IND $= 1$, if IND on input was 0 or 1, or IND $= 11$, if IND on input was 10 or 11.

If the code within PDEDEF cannot be parallelized, you must **not** add any OpenMP directives to your code, and must **not** set IND to 10 or 11. If IND is set to 10 or 11 and PDEDEF has not been parallelized, results on multiple threads will be unpredictable and may give rise to incorrect results and/or program crashes or deadlocks. Please contact NAG for advice if required. Overloading IND in this manner is not entirely satisfactory, consequently it is likely that replacement interfaces for D03RAF and D03RBF will be included in a future NAG Library release.

Modified example programs for D03RAF and D03RBF, which include parallel versions of the PDEDEF routines, are included in the distribution material for each implementation of the NAG Library for SMP & Multicore.

2.2.4 Global Optimization of a Function (Chapter E05)

Users of the Particle Swarm Optimization (PSO) routines in the NAG Library for SMP & Multicore need to be aware of several additional SMP-specific options they need to set, in particular, to state whether or not they have ensured that the user functions they provide to these routines are implemented in a thread-safe manner. See the routine documents for E05SAF and E05SBF for details.

2.2.5 Sparse Iterative Solvers (Chapter F11)

When running the sparse iterative solvers with preconditioning on multiple processors, it may be beneficial to reduce the action of the preconditioner, e.g., by decreasing LFILL, or by increasing DTOL with LFILL < 0 in F11DAF or F11JAF. This will tend to increase the number of iterations required to obtain a converged solution, but will also allow a greater percentage of the computational work to be spent in the parallelized iterative solvers, resulting in a lower overall time to solution. There is unfortunately no choice of the various preconditioner parameters which is optimal for all types of matrix, and all numbers of processors, and some experimentation will generally be required for each new type of matrix encountered.

2.2.6 Quasi-random number generators (Chapter G05)

The Sobol, Sobol (A659) and Niederreiter quasi-random number generators in G05YMF have been parallelized, but require quite large problem sizes to see any significant performance gain. Parallelism is only enabled for the RCORD $= 2$ option.

3 References

OpenMP *The OpenMP Specification for Parallel Programming* http://www.openmp.org