DGEFMM User's Guide

DGEFMM version 1.0 November 12, 1996

A Portable Implementation of Strassen's Algorithm

Steven Huss-Lederman[†]
Elaine M. Jacobson[‡]
Jeremy R. Johnson[*]
Anna Tsao[‡]
Thomas Turnbull[‡]

[†]Computer Sciences Department
University of Wisconsin – Madison
Madison, WI 53706

[‡]Center for Computing Sciences
Bowie, Maryland 20715

[*]Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104

Steven Huss-Lederman
lederman@cs.wisc.edu

Elaine M. Jacobson
emj@super.org

Jeremy R. Johnson
jjohnson@king.mcs.drexel.edu

Anna Tsao
anna@super.org

Tom Turnbull
turnbull@super.org

ACKNOWLEDGMENTS

This work is partially supported by DARPA's Applied and Computational Mathematics Program under order number C65.

CONTACT

Please feel free to send questions, comments, and problem reports to `prism@super.org`.

Contents

1 Introduction

The PRISM project is a joint effort of the Argonne National Laboratory, Center for Computing Sciences, Duke University, University of California at Berkeley, University of Kentucky, University of Texas at Austin, and University of Wisconsin – Madison. Its overall goal is the development of scalable, parallel eigensolvers for distributed memory systems.

The software discussed in this document relates to the package developed for multiplying matrices using Strassen's algorithm which is joint work between Center for Computing Sciences, Drexel University, and University of Wisconsin – Madison.

This package, known as DGEFMM, uses Winograd's variant of Strassen's algorithm for multiplying matrices of arbitrary size. The advantage of using a Strassen-based algorithm is that, for square matrices, it has a scaling of $O(n^{lg(7)})$ instead of the conventional algorithm in DGEMM which scales as $O(n^3)$, where n is the order of the matrix, and $lg(7)$ is the base 2 logarithm of 7, which is approximately 2.809. The name was chosen to represent the fact that this algorithm is often faster than DGEMM (FMM stands for Fast Matrix Multiplication). It has the same functionality as DGEMM which computes $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$, where α and β are scalars, and $\text{op}(X)$ can either be X or X^T (transpose of X).

Some of the main advantages of our implementation are:

- The routines are "plug-and-play" with the Level 3 BLAS routine DGEMM for matrix multiplication. Thus, users can use our code with minimal effort. The code automatically uses the Level 3 BLAS DGEMM routine for smaller matrices when it is faster to do so. Thus, using our routine should always be as fast or faster than using DGEMM alone.
- Our routine often uses less memory than previous Strassen-based matrix multiplication routines. For example, for square matrices, it needs $2/3n^2$ to n^2 work space depending on the input parameters.
- Improved cutoff criteria for rectangular matrices.
- The code is written in C, with limited Fortran routines used on some systems. Consequently, it is portable across a wide variety of machines. Both C and Fortran interfaces are available. To be compatible with the Level 3 BLAS routine DGEMM, column major storage (Fortran style) is used for all 2-dimensional arrays.
- All the source code is publicly available and comes free of charge.

See Section 6 for obtaining papers describing our research in this area as well as details of this library.

All computations are in 64-bit precision only. Users interested in other precisions should contact `prism@super.org`.

2 Portability

2.1 BLAS

The DGEFMM routines utilize the Level 3 BLAS DGEMM routine. On many machines a fast vendor-supplied version of this routine is provided in their BLAS library. Portable versions of this library can be obtained from `netlib`. The World Wide Web URL for netlib is

`http://www.netlib.org/index.html`.

Anonymous ftp access is available through the server at `ftp.netlib.org`.

2.2 Optimization

The machine-dependent parameters for choosing when to use DGEMM instead of applying further Strassen recursions is given in the header file `matrix.h`. The machines listed in the supported systems section have values provided. These should be reasonable choices for these machines. For other machines the values are set to preset defaults. Those installing the package can find the optimal choice for these parameters via the methods discussed in our papers.

2.3 Makefile

The software is built using make. The Makefile will call the appropriate compilers and linkers based on make include files which define these machine-dependent features. See Section 3 for further details.

2.4 Supported Systems

The DGEFMM software should run on any machine. The software is distributed in a tar file so you will also need tar. The software has been tested and run on Cray vector machines (C90), RS/6000 machines (workstations and an SP node), Sun Workstations, and a node of the T3D.

2.5 Interfacing Fortran to C

We offer both Fortran and C interfaces to our routines. Our main routines, however, are written in C. Thus, the Fortran interfaces are wrapper routines written in C that accept Fortran style arguments. There is great variation in how systems take the Fortran routine name and create the link name. For example, some systems only use uppercase while others append an underscore. What is done with C routines is often different. We have attempted to have the proper C name used in our Fortran interface routines so that linking will occur without any problems. However, there may still be problems on some systems. Resolution may require editing the include file, `matrix.h`, to translate the routine name. You are also welcome to contact us for further assistance. We also welcome feedback on necessary changes so we can incorporate them into future releases to avoid the necessity of others having to edit our files.

3 Installation

3.1 Obtaining DGEFMM

DGEFMM, a serial implementation of Winograd's variant of Strassen's algorithm, is distributed as portable ANSI C source code with limited Fortran routines in a single tar file. This distribution file contains all the needed files. The latest distribution can be obtained from the PRISM home page at `http://www.mcs.anl.gov/Projects/PRISM`. The latest version of the entire DGEFMM library is available on the page `http://www.mcs.anl.gov/Projects/PRISM/lib/software.html` by choosing `prism_dgefmm.tar`.

Create a directory to hold the DGEFMM code, for example `~/DGEFMM`, and place the distribution file in that directory. The distribution file can then be untarred with the UNIX command

```
tar xf prism_dgefmm.tar
```

3.2 Setting Environment Variables

The compilation of the library is done via a Makefile. This should work automatically once two environment variables are set. The environment variables are necessary to tell the Makefile how to link in the BLAS and the architecture for which you are compiling. The environment variables are:

`PRISM_MPI_ARCH` : Machine architecture. Supported values are: `cray` for Cray vector machines, `cray_t3d` for the Cray T3D, `rs6000` for IBM RS/6000 class machines, `solaris` for Sun Sparc architectures running solaris, and `sun4` for Sun Sparc architectures running older SunOS. The Makefile will include file Make-<PRISM_MPI_ARCH>.include to define the machine-specific parameters.

`PRISM_BLAS_LIB` : Linker flag needed to link in the BLAS library. Some vendors supply a BLAS library. On the `rs6000` it is normally set to `-lblas -lxlf90`; both `cray` and `cray_t3d` use `-lsci`. Since the BLAS are often written in Fortran, you may need to add in linker flags to include the Fortran libraries.

If you have already set up your environment for other PRISM codes then these are just a subset of the same variables as before. The names were left the same to make it easier for users of our other codes. Note that many of our codes are designed for parallel machines and this is why the `PRISM_MPI_ARCH` name has MPI in it. However, this serial code does **NOT** use MPI and is **NOT** parallel. It can be (and has been) run serially on nodes of a parallel machine.

The values of `PRISM_MPI_ARCH` given above are the ones for which we supply the make include file. In general, it is very easy to create a new `PRISM_MPI_ARCH`. To do this, decide what the new value of `PRISM_MPI_ARCH` will be. For example, suppose you are going to make it `newarch`. Then copy one of the Make include files (for example the sun4 one) by doing:

cd make_include/

cp Make-sun4.include Make-newarch.include

Edit the `Make-newarch.include` file to define the machine-specific information. You will need to supply the name of the C and Fortran compiler, optimization flags, and similar values.

3.3 Building the DGEFMM Library

To build the DGEFMM library and test code, set the `PRISM_*` environment variables as described in Section 3.2, `cd code/` and then type `make`. By default, both the library and test code will be made. To only make the library version use `make stras_lib`. To only make the testing code use `make stras_testing`. Use of the library is described in Section 4 and running the test code is described in Section 3.4.

To keep objects, executables and libraries separate for each `PRISM_MPI_ARCH`, the value in `PRISM_MPI_ARCH` is added to each filename. This allows one to create different architectures in the same directory without cleaning up previous files. Thus, the files are named:

routine.o_⟨`PRISM_MPI_ARCH`⟩: object file for program routine

stras_lib_⟨`PRISM_MPI_ARCH`⟩.a: DGEFMM library

timefmm_⟨`PRISM_MPI_ARCH`⟩: test program

The library is placed in the lib/ subdirectory.

During the building of the DGEFMM library, some machines will issue warning messages which refer to non-standard object file name extensions. These are due to our use of different object file extensions for different `PRISM_MPI_ARCH`. An example of such a message on the RS/6000 is:

```
xlc:  1501-218 file fmm_internal.o_rs6000 contains an incorrect file suffix
```

These warning can be safely ignored.

There are several maintenance options for the Makefile:

cleanobjs: removes all object files for the current `PRISM_MPI_ARCH`.

clean: removes all object files, emacs backup files (*~), and the test code for the current `PRISM_MPI_ARCH`.

3.4 Testing

The easiest way to test the code is to use the script `test_suite`. It will automatically create an appropriate testing version of the code and run 10 random rectangular matrices up to size 1000. At the end, script will normally output

```
No errors were detected and all tests completed.
```

to indicate no errors occurred. If errors do occur, they will be noted to the screen. Most output is redirected to the file out_test_⟨`PRISM_MPI_ARCH`⟩. A sample output can be found in `out_test_sample`. This file will contain comparison timing between DGEMM and DGEFMM.

The test script will remove all the objects for the current architecture during the making of the test code. This should not affect anything since all executables and libraries are left intact.

Normally, the whole test script will run in a few minutes. On some systems with a very slow DGEMM routine, the time can be longer. For example, the current DGEMM routine distributed with the BLAS has poor blocking features on Suns. This means that running DGEMM can be very slow. As a result, running `test_suite` took over 10 minutes on a Sparc 10.

In general, this is all the testing that is needed. Some users may want to run additional tests and timing runs. For these users, they can write their own interface routine or use our routine. The make in Section 3.3 will create the executable timefmm_⟨`PRISM_MPI_ARCH`⟩. This code can test and time DGEMM and DGEFMM. It expects several inputs. If not provided you will be prompted for them. The inputs are:

m n k: The three matrix dimensions. Matrix A will have dimension $m \times k$, matrix B will have dimension $k \times n$, and matrix C will have dimension $m \times n$.

square cutoff: The order at which to stop recursions for square matrices. If you enter a value of zero or less then the default value will be used.

cutoff_m: The size at which to stop recursions in the m-dimension for rectangualar matrices. If you enter a value of zero or less then the default value will be used.

cutoff_n: The size at which to stop recursions in the n-dimension for rectangualar matrices. If you enter a value of zero or less then the default value will be used.

cutoff_k: The size at which to stop recursions in the k-dimension for rectangualar matrices. If you enter a value of zero or less then the default value will be used.

DGEMM?: If you do **NOT** want to compare to DGEMM then enter 1. Any other value will result in a comparison to DGEMM.

Only the first three input parameters are required; all other parameters will use default values if they are not provided. Default values can be found in file `matrix.h`.

3.5 Description of Software Provided

The software tree is:

`doc/` **:** Contains a PostScript version of this file as well as any other documentation. It also has the COPYRIGHT notice. See Section 6 for obtaining papers on these routines and our research in this area.

`code/` **:** Contains the source code, Makefile, test script, and sample output. These are, in alphabetical order:

- **DGEFMM.c:** DGEMM look-alike interface for fmm.c. This is a FORTRAN interface which mallocs all necesarry temporary space internally.
- **DGEFMM_MEM.c:** Similar to DGEFMM.c but adds two parameters to allow user to provide temporary space.
- **fixup_internal.c:** Performs cleanup of remaining row(s)/col(s)
- **fmm.c:** Main driver program. This is the same as DGEFMM_MEM.c but with a C interface.
- **fmm_internal.c:** Fast Matrix Multiply on arbitrary-sized matrices
- **gen_matrix.c:** Generates test matrices
- **matrix.c:** Matrix utility routines
- **matrix_arith.F:** Matrix arithmetic routines
- **out_test_sample:** Sample output of results from running `test_suite`
- **prism_time.c:** Elapsed timer for use on some systems used in testing
- **strassen_internal.c:** Fast Matrix Multiply on even-sized matrices
- **test_suite:** Script for running test suite
- **timefmm.c:** Test code

`include/` **:** Contains the header files for C. These are:

- **matrix.h:** Include file for main routines
- **matrix_test.h:** Include file for testing routines

`lib/` **:** Not included in the distribution; will contain the libraries built after make has completed.

`make_include/` **:** Contains the files included into the Makefile to define machine-dependent parameters. The file choice is based on an environment variable that defines the architecture as described in Section 3.2. The files have the syntax:

`Make-`<PRISM_MPI_ARCH>`.include`

where we currently have files for <PRISM_MPI_ARCH> of:

cray: for Cray vector machines

cray_t3d: for Cray T3D

rs6000: for RS/6000 machines, including SP

solaris: for Sun workstations running Solaris

sun4: for Sun workstations running SunOS

`man/manl/` : Contains man pages for our main routines as well as the standard dgemm man page.

3.6 CPP Flags

Typically, the code built will meet your needs if you simply use the default values supplied. However, if you need to install the code on an architecture for which a make include file is not provided, or you want to do specific testing and timings, you will need to modify the default CPP options.

The CPP options for modifying the behavior of the test code are:

`STRAS_CRAY`: Runs the Cray library routine DGEMMS for comparison to our routine. It can only be used if you are working on a Cray machine (PRISM_CRAY is set).

`STRAS_ESSL`: Runs the IBM ESSL library routine DGEMMS for comparison to our routine. It can only be used if you are working on an IBM machine (PRISM_SP is set). The ESSL library must be available.

`STRAS_NON_SQ`: Takes input and runs non-square test matrices.

`STRAS_RANDOM`: Allows for generation of random test matrices.

The following CPP options are normally set in the make include files:

`PRISM_CRAY`: Chooses code specific to Cray vector machines.

`STRAS_BLAS_ADD`: Uses a combination of BLAS calls to do matrix addition. Not on by default because it requires extra arithmetic.

`STRAS_BLAS_SUB`: Uses a combination of BLAS calls to do matrix subtraction. Not on by default because it requires extra arithmetic.

`STRAS_ESSL_ADD`: Uses the IBM ESSL library routine DGEADD to do matrix addition. Not on by default because it requires ESSL and does not significantly change the performance of the code.

`STRAS_ESSL_SUB`: Uses the IBM ESSL library routine DGESUB to do matrix subtraction. Not on by default because it requires ESSL and does not significantly change the performance of the code.

`STRAS_FORTRAN_ADD`: Uses Fortran code to do matrix addition. This can be faster on some machines.

`STRAS_FORTRAN_SUB`: Uses Fortran code to do matrix subtraction. This can be faster on some machines.

`PRISM_SP`: Chooses code specific to IBM RS/6000 machines. It has `SP` because our parallel codes use the RS/6000 processor on the SP machines. Note, again, that this code is serial.

`PRISM_SUN`: Chooses code specific to Sun workstation machines (SunOS and Solaris).

`PRISM_T3D`: Chooses code specific to Cray T3D machines.

There are several other CPP options that should only be used, in general, for our internal development. They may be removed in the future when the final cleanup of the code occurs.

4 Using the DGEFMM Routine

Once you have built the library as described in Section 3.3, you can use it to link in the needed routines. We provide three routines/entry points that are normally used. All of these routines run the same core routines. The different calling mechanisms only differ in whether they are C or Fortran and whether the user supplies the temporary workspace. All arrays are stored in column-major format as in Fortran. This allows our routines to have the same input as the Level 3 BLAS routine DGEMM.

Fortran users have two interface routines to call. The first has the identical calling sequence to the Level 3 BLAS routine DGEMM. It automatically mallocs the necessary temporary space inside the library. The calling sequence is:

DGEFMM(C_TRANSA, C_TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

A description of the calling sequence can be found in the man page for DGEMM which is given in Section 9. The second Fortran interface, which allows the user to supply the needed temporary space, is:

DGEFMM_MEM(C_TRANSA, C_TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C,
LDC, WORK, LWORK)

where two parameters were added at the end. These are:

WORK: Array containing space for use by the library. On exit, its values have been modified since this is used as temporary space internally. Thus, the values are not significant on exit.

LWORK: The number of 64-bit precision words in array **WORK**. The code's memory usage depends on the number of levels of recursion performed. Let r be the number of levels of recursion. Except when r is a very small integer, an upper bound on the memory usage which is close to the actual usage can be obtained by assuming $r = \infty$. The current code uses

$$\lceil 1/3\,(1-(1/4)^r)\,(mk+kn+mn)\rceil \quad \text{when } \beta \neq 0$$

and

$$\lceil 1/3\,(1-(1/4)^r)\,(m \times max(k,n)+kn)\rceil \quad \text{when } \beta = 0$$

where $\lceil x \rceil$ is the smallest integer greater than x. Note that if DGEMM is faster than DGEFMM then no recursions are performed ($r = 0$). As the formulas indicate, no temporary memory is used in this case.

A routine with the same calling sequence, but with a C interface (call by value instead of reference) is:

Matrix Order: Number of Recursions (r)	RS/6000 DGEMM	RS/6000 DGEFMM	C90 DGEMM	C90 DGEFMM	T3D DGEMM	T3D DGEFMM
$\tau + 1:\ 1$	.150	.150	.0060	.0055	.694	.669
$2\tau + 2:\ 2$	1.14	1.05	.0431	.0410	5.40	4.91
$4\tau + 4:\ 3$	9.06	7.59	.332	.312	42.6	33.3
$8\tau + 8:\ 4$	72.2	54.1	2.54	2.10		
$16\tau + 16:\ 5$			20.1	13.3		

Table 1: *Times in seconds for DGEMM and DGEFMM for different matrix sizes on three machines with $\alpha = 1/3$ and $\beta = 1/4$.*

fmm(char transa, char transb, int m, int n, int k, double alpha, double *a, int lda, double *b, int ldb, double beta, double *c, int ldc, double *work, int lwork)

If **lwork** is too small or **work** equals NULL, then the routine mallocs the necessary temporary space needed.

The user can also call the routine we use to compute the amount of temporary memory needed. The Fortran interface is:

INTEGER FUNCTION TMP_DGEFMM_MEM(M, N, K, ALPHA, BETA)

and the C interface:

int tmp_fmm(int m, int n, int k, double alpha, double beta)

The parameters are as described in the calls above. The routine returns the number of 64-bit precision words of temporary space needed.

5 Performance

The performance of DGEFMM relative to DGEMM varies from machine to machine and varies depending on matrix size. The best information on performance can be found in our papers. Table 1 gives some idea of the relative performance DGEFMM to DGEMM for 3 machines and various square matrix sizes. The order of the matrix for each machine can be determined by substituting for τ the values 199, 129, and 325 for the RS/6000, C90, and T3D, respectively. For example, on the T3D, for a square matrix of order $4(325) + 4 = 1304$, timings are 42.6 and 33.3 seconds for DGEMM and DGEFMM, respectively. All timing given are when the matrices and work space fit in core memory. These routines have not been optimized for the out-of-core case and would be expected to run substantially slower.

6 Papers

This work was done as part of the PRISM project which has a home page at `http://www.mcs.anl.gov/Projects/PRISM`. Papers describing our research as well as details about this library are available from `http://www.mcs.anl.gov/Projects/PRISM/lib/tech.html`. Choose the option to see papers on Strassen. These papers have a number of plots indicating the performance of our routines on various machines.

7 Bugs

There are no known bugs at this time.

8 Contacts

Comments, questions, bugs and requests should be sent to `prism@super.org`. We welcome all input.

9 Manual Pages

The directory `man/manl/` has man pages for all the user visible routines. The man page for DGEFMM is included here for reference.

```
DGEFMM(l)          PRISM routine (version 1.0)          DGEFMM(l)

PRISM routine     Last change: 12 November 1996                 1

NAME
     DGEFMM - Winograd's variant of Strassen's algorithm to  per-
     form one of the matrix-matrix operations
     C := alpha*op( A )*op( B ) + beta*C.

SYNOPSIS
     SUBROUTINE DGEFMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA,
                         B, LDB, BETA, C, LDC )

         CHARACTER*1       TRANSA, TRANSB

         INTEGER           M, N, K, LDA, LDB, LDC

         DOUBLE PRECISION  ALPHA, BETA

         DOUBLE PRECISION  A( LDA, * ), B( LDB, * ), C( LDC, * )

PURPOSE
     DGEFMM  performs one of the matrix-matrix operations
```

```
     C := alpha*op( A )*op( B ) + beta*C,

     where  op( X ) is one of

        op( X ) = X   or   op( X ) = X' (transpose of X),

     alpha and beta are scalars, and A, B  and  C  are  matrices,
     with  op(  A  ) an m by k matrix,  op( B )  a  k by n matrix
     and  C an m by n matrix.

     DGEFMM is designed to be plug-and-play with the Level 3 BLAS
     routine  DGEMM.   It  uses  Winograd's variant of Strassen's
     algorithm to reduce the operations count relative to conven-
     tional  matrix  multiplication  (DGEMM).   It  has a Fortran
     interface.  All arrays are stored in  column  major  format.
     The  necessary  temporary  memory  is  dynamically  malloced
     inside the routine.


PARAMETERS
     TRANSA  - CHARACTER*1.
             On entry, TRANSA specifies the form of op( A ) to be
             used in the matrix multiplication as follows:

             TRANSA = 'N' or 'n',  op( A ) = A.

             TRANSA = 'T' or 't',  op( A ) = A'.

             TRANSA = 'C' or 'c',  op( A ) = A'.

             Unchanged on exit.

     TRANSB  - CHARACTER*1.
             On entry, TRANSB specifies the form of op( B ) to be
             used in the matrix multiplication as follows:

             TRANSB = 'N' or 'n',  op( B ) = B.

             TRANSB = 'T' or 't',  op( B ) = B'.

             TRANSB = 'C' or 'c',  op( B ) = B'.

             Unchanged on exit.

     M       - INTEGER.
             On entry,  M  specifies  the number  of rows  of the
             matrix  op( A )  and of the  matrix  C.  M  must  be
             at least  zero.  Unchanged on exit.
```

```
N      - INTEGER.
         On entry,  N  specifies the number  of columns of
         the matrix op( B ) and the number of columns of the
         matrix C. N must be at least zero.  Unchanged on
         exit.

K      - INTEGER.
         On entry,  K  specifies the number of columns of
         the matrix op( A ) and the number of rows of the
         matrix op( B ). K must be at least zero. Unchanged
         on exit.

ALPHA  - DOUBLE PRECISION.
         On entry, ALPHA specifies the scalar alpha.
         Unchanged on exit.

A      - DOUBLE PRECISION array of DIMENSION ( LDA, ka ),
         where ka is k  when  TRANSA = 'N' or 'n',  and is  m
         otherwise.  Before entry with  TRANSA = 'N' or 'n',
         the leading  m by k part of the array  A  must con-
         tain the matrix  A,  otherwise the leading  k by m
         part of the array  A  must contain  the matrix  A.
         Unchanged on exit.

LDA    - INTEGER.
         On entry, LDA specifies the first dimension of A  as
         declared  in the calling (sub) program. When  TRANSA
         = 'N' or 'n' then LDA must be at least  max( 1, m ),
         otherwise  LDA must be at least  max( 1, k ).
         Unchanged on exit.

B      - DOUBLE PRECISION array of DIMENSION ( LDB, kb ),
         where kb is n  when  TRANSB = 'N' or 'n',  and is  k
         otherwise.  Before entry with  TRANSB = 'N' or 'n',
         the leading  k by n  part of the array  B  must
         contain  the matrix  B,  otherwise the leading  n by
         k  part of the array  B  must contain  the matrix B.
         Unchanged on exit.

LDB    - INTEGER.
         On entry, LDB specifies the first dimension of B  as
         declared  in the calling (sub) program. When  TRANSB
         = 'N' or 'n' then LDB must be at least  max( 1, k ),
         otherwise  LDB must be at least  max( 1, n ).
         Unchanged on exit.

BETA   - DOUBLE PRECISION.
```

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input. Unchanged on exit.

C - DOUBLE PRECISION array of DIMENSION (LDC, n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).

LDC - INTEGER. On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m). Unchanged on exit.

SEE ALSO

DGEFMM_MEM(l), fmm(l), TMP_DGEMM_MEM(l), tmp_fmm(l).

FURTHER INFORMATION

See the DGEFMM User's Guide provided in the doc/ directory of the distribution.

BUGS

There are no known bugs.

COPYRIGHT

See COPYRIGHT file provided with the software distribution.

AUTHORS

Steven Huss-Lederman, University of Wisconsin
Elaine M. Jacobson, Center for Computing Sciences
Jeremy R. Johnson, Drexel University
Anna Tsao, Center for Computing Sciences
Thomas Turnbull, Center for Computing Sciences