April 1991
Order Number: 312128-001

iPSC®/860
BASIC MATH LIBRARY
USER'S GUIDE

intel® Corporation

REV.	REVISION HISTORY	DATE
-001	Original Issue	04/91

RESTRICTED RIGHTS

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the rights in Technical Data and Computer Software clause at 52.227-7013. Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051.

PREFACE

This manual describes the CLASSPACK Basic Math Library from Kuck and Associates for the iPSC®/860 system.

This manual assumes that you are an application programmer proficient in the C and Fortran languages and the UNIX operating system.

ORGANIZATION

Chapter 1	"Overview", introduces the BLAS and Fast Fourier Transform routines, and describes how to use them.
Chapter 2	"Routine Descriptions" provides detailed descriptions of the routines.
Appendix A	"Routine Arguments" describes how vector and matrix arguments are passed.
Appendix B	"Examples" contains code segments that illustrate the calling sequence of programs in the Basic Math Library.
Appendix C	"Performance Hints" presents suggestions on how to obtain optimal performance from the routines in the Basic Math Library on systems based on the Intel i860 processor.
Appendix D	"Performance Evaluation of the BLAS Math Library" describes the tests used to evaluate performance on the iPSC system, and contains the resulting graphs and values for each routine.

APPLICABLE DOCUMENTS

iPSC® System Manuals

iPSC®/2 and iPSC®/860 Interactive Parallel Debugger Manual
Tells how to use the iPSC Interactive Parallel Debugger and provides command reference information.

iPSC®/2 and iPSC®/860 Math Libraries Reference Manual
Describes the math libraries available on the iPSC system.

iPSC®/2 and iPSC®/860 Programmer's Reference Manual
Describes iPSC system commands and system calls (both C and Fortran).

iPSC®/2 and iPSC®/860 User's Guide
Overviews the iPSC system, including hardware and software architectures. Tells how to develop and run programs.

iPSC®/860 C Compiler User's Guide
Tells how to use the iPSC/860 C compiler driver.

iPSC®/860 Fortran Compiler User's Guide
Tells how to use the iPSC/860 Fortran compiler driver.

Intel® Manuals

UNIX System V Manual Set
Describes UNIX System V.

i860™ Manual Set
Describes the i860 microprocessor.

Other Manuals

C: A Reference Manual - Harbison and Steele
Describes the C programming language.

The C Programming Language - Kernighan and Ritchie
Describes the C programming language.

CLASSPACK

Basic Math Library

User's Guide

Release 1.1

Document #9103001
by Kuck & Associates, Inc., Champaign, IL 61820

CLASSPACK
Basic Math Library
User's Guide

Release 1.1

March, 1991

by Kuck & Associates, In., Champaign, IL 61820

Printed in the United States of America.

Table of Contents

Chapter 1

Overview

1.1. Introduction

Kuck and Associates' CLASSPACK *Basic Math Library* contains highly optimized versions of standard computational building blocks which users of high-performance computers can use to improve the speed and portability of their numerical applications programs.

The other optimized libraries in the CLASSPACK family are the Dense Matrix, Sparse Matrix, Out-of-Core Solver, and Signal Processing Libraries.

1.1.1. The BLAS

The BLAS (*Basic Linear Algebra Subprograms*) routines, which make up the bulk of this library, are widely used in dense numerical linear algebra programs. Since they are the most useful vector and matrix operations, the availability of versions tuned to the user's machine provides a simple way for speeding up existing programs and writing new numerical programs. They also enhance portability, since programs can contain standard calls to a widely-available set of library routines.

The BLAS routines were developed in three groups. The *BLAS 1* routines consist of vector-vector operations such as dot products and "scalar*vector + vector". The *BLAS 2* routines provide matrix-vector operations such as matrix-vector multiply and the rank-1 update of a matrix. The *BLAS 3* routines provide matrix-matrix operations such as rank-k update and the solution of triangular systems with multiple right-hand sides. This User's Guide does not separate the different BLAS levels.

1.1.2. Fast Fourier Transforms

The *Fourier Transform* is a common operation in many fields, especially signal processing. Kuck and Associates' *Basic Math Library* includes three routines (in single-precision and double-precision versions) for performing FFTs (*Fast Fourier Transforms*). These have been tuned for optimal performance on this machine.

While FFT routines are not as standardized as the BLAS routines, portability is eased by limiting changes to small sections of code which interface between the user's program and the available library.

1.2. Using the Basic Math Library

The *Basic Math Library* object routines are packaged as a library archive named *libkmath.a*. Programs can be linked with routines in this library by using the `-lkmath` option on the Fortran compiler command line.

The next section, and the appendices, provide information to aid the user in making the most effective use of this optimized library.

1.2.1. Argument Checking

The descriptions of the routines in this manual specify the legal values for all scalar integer and character arguments. The routines themselves check these arguments when they are called to determine if they have legal values. If the value of one of the arguments is not legal, an error action is taken. The action taken depends on the class of routine.

- BLAS Level 1 Routines: Return from the program without performing any computations.

- BLAS Level 2 Routines: Issue an error message which contains the name of the routine and the number of the argument with the illegal value, e.g.,

```
 ** On entry to DGEMV  parameter number  1 had an illegal
value
```

- BLAS Level 3 Routines: Issue an error message which contains the name of the routine and the number of the argument with the illegal value.

```
 ** On entry to DGEMM  parameter number  1 had an illegal
value
```

- Fast Fourier Transforms: return from the program without performing any computations.

Chapter 2

Routine Descriptions

2.1. BLAS routines

The following sections describe the BLAS (Basic Linear Algebra Subprograms) routines available in the *Basic Math Library*.

The routines are arranged in alphabetical order, ignoring the prefixes which indicate the data type the specific routine uses. (E. g., routine names beginning with **D** are **DOUBLE PRECISION**.) All of the BLAS routines come in multiple versions, usually for SINGLE PRECISION real, DOUBLE PRECISION real, single precision COMPLEX, and DOUBLE PRECISION COMPLEX.

For more information, consult the following articles:

C. Lawson, R. Hanson, D. Kincaid, and F. Krough, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Trans. on Math. Soft.* 5 (1979) 308-325.

J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Trans. on Math. Soft.* 14,1 (1988) 1-32.

J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans on Math. Soft.* (Dec. 1989).

isamax/idamax/icamax/izamax

Syntax

iw = isamax/idamax/icamax/izamax (n, x, incx)

Purpose

isamax/idamax finds the smallest index *i* such that:

$$|x_i| = \max(|x_j|), \quad j = 1 .. n$$

icamax/izamax finds the smallest index *i* such that:

$$|\mathrm{Re}(x_i)| + |\mathrm{Im}(x_i)| =$$
$$\max(|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)|), \quad j = 1 .. n$$

(Note that **icamax** and **izamax** use the sum of the magnitudes of the real and imaginary parts of x_j, not the magnitude of the complex value.)

On Entry

n integer

the order of vector *x*.

x Single precision real for isamax
Double precision real for idamax
Single precision complex for icamax
Double precision complex for izamax

array of dimension *idimx*
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*. This is the distance in array x between elements of vector *x*. See Appendix A for notes on allowed values of spacing increments.

On Return

iw integer

index of largest (absolute value) element of *x*.

sasum/dasum/scasum/dzasum

Syntax

w = sasum/dasum/scasum/dzasum (n, x, incx)

Purpose

sasum/dasum/scasum/dzasum computes the sum of magnitudes of the elements of a vector:

$$\sum |x_i|$$

On Entry

n integer

the order of vector *x*.

x Single precision real for sasum
Double precision real for dasum
Single precision complex for scasum
Double precision complex for dzasum

array of dimension *idimx*
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x* (i.e., the distance in array *x* between elements to be included in the summation).

On Return

w Single precision real for sasum
Double precision real for dasum
Single precision real for scasum
Double precision real for dzasum

sum of magnitudes of elements of *x*.

saxpy/daxpy/caxpy/zaxpy

Syntax

call saxpy/daxpy/caxpy/zaxpy (n, alpha, x, incx, y, incy)

Purpose

saxpy/daxpy/caxpy/zaxpy computes a vector-scalar product:

$$y = \alpha x + y$$

where α is a scalar and x and y are vectors.

On Entry

n integer

the order of vectors x and y.

alpha Single precision real for saxpy
Double precision real for daxpy
Single precision complex for caxpy
Double precision complex for zaxpy

scalar α.

x Single precision real for saxpy
Double precision real for daxpy
Single precision complex for caxpy
Double precision complex for zaxpy
array of dimension *idimx*

contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*.

y Single precision real for saxpy
Double precision real for daxpy
Single precision complex for caxpy
Double precision complex for zaxpy
array of dimension *idimy*

contains the spaced vector *y* of order *n*. The dimension *idimy* must be at least as large as `1+|incy|*(n-1)`.

incy integer

spacing increment for vector y.

On Return

y contains the spaced vector y of order n.

scopy/dcopy/ccopy/zcopy

Syntax

call scopy/dcopy/ccopy/zcopy (n, x, incx, y, incy)

Purpose

scopy/dcopy/ccopy/zcopy performs the copying operation:

$$y = x$$

where x and y are n-element vectors.

On Entry

n integer

the order of vectors *x* and *y*.

x Single precision real for scopy
Double precision real for dcopy
Single precision complex for ccopy
Double precision complex for zcopy

array of dimension *idimx*
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector x.

y Single precision real for scopy
Double precision real for dcopy
Single precision complex for ccopy
Double precision complex for zcopy

array of dimension *idimy*
contains the spaced vector *y* of order *n*. The dimension *idimy* must be at least as large as `1+|incy|*(n-1)`. *y* does not need to be defined on entry.

incy integer

spacing increment for vector y.

On Return

y	contains the spaced vector *y* of order *n*.

sdot/ddot/dsdot

Syntax

w = sdot/ddot/dsdot (n, x, incx, y, incy)

Purpose

sdot/ddot/dsdot computes the dot product of two vectors:

$$x^T y$$

sdot is entirely single precision.
ddot is entirely double precision.
dsdot takes single precision arguments, but performs the summation in double precision and returns a double precision result.

On Entry

n integer

the order of vectors x and y.

x Single precision real for sdot
Double precision real for ddot
Single precision real for dsdot

array of dimension idimx
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*.

y Single precision real for sdot
Double precision real for ddot
Single precision real for dsdot
array of dimension *idimy*.
Contains the spaced vector *y* of order *n*. The dimension *idimy* must be at least as large as `1+|incy|*(n-1)`.

incy integer

spacing increment for vector *y*.

On Return

w	Single precision real for sdot Double precision real for ddot Double precision real for dsdot the dot product of x and y.

cdotc/zdotc

Syntax

w = cdotc/zdotc (n, x, incx, y, incy)

Purpose

cdotc/zdotc computes the dot product of a (conjugated) vector and another vector:

$$x^H y$$

where x and y are n-element vectors.

On Entry

n integer

the order of vectors x and y.

x Single precision complex for cdotc
Double precision complex for zdotc

array of dimension *idimx*.
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*.

y Single precision complex for cdotc
Double precision complex for zdotc

array of dimension *idimy*.
Contains the spaced vector *y* of order *n*. The dimension idimy must be at least as large as `1+|incy|*(n-1)`.

incy integer

spacing increment for vector *y*.

On Return

w	Single precision complex for cdotc Double precision complex for zdotc the dot product of the (conjugated) *x* and (unconjugated) *y*.

cdotu/zdotu

Syntax

w = cdotu/zdotu (n, x, incx, y, incy)

Purpose

cdotu/zdotu computes the dot product of two vectors:

$$x^T y$$

where x and y are n-element vectors.

On Entry

n integer

the order of vectors x and y.

x Single precision complex for cdotu
Double precision complex for zdotu

array of dimension *idimx*.
Contains the spaced vector x of order n. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector x.

y Single precision complex for cdotu
Double precision complex for zdotu

array of dimension *idimy*.
Contains the spaced vector y of order n. The dimension *idimy* must be at least as large as `1+|incy|*(n-1)`.

incy integer

spacing increment for vector y.

On Return

w	Single precision complex for cdotu Double precision complex for zdotu the dot product of x and y.

sdsdot

Syntax

w = sdsdot (n, alpha, x, incx, y, incy)

Purpose

sdsdot adds a constant to the dot product of two vectors:

$$\alpha + x^T y$$

sdsdot takes single precision arguments, performs the summation in double precision, and returns a single precision result.

On Entry

n integer

the order of vectors x and y.

alpha single precision real

the additive constant α.

x single precision real

array of dimension idimx
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*.

y single precision real

array of dimension *idimy*.
Contains the spaced vector *y* of order *n*. The dimension *idimy* must be at least as large as `1+|incy|*(n-1)`.

incy integer

spacing increment for vector *y*.

On Return

w	single precision real	
	the dot product of x and y.	

sgbmv/dgbmv/cgbmv/zgbmv

Syntax

call sgbmv/dgbmv/cgbmv/zgbmv (trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy)

Purpose

sgbmv/dgbmv/cgbmv/zgbmv performs one of the matrix-vector operations

$$y=\alpha Ax+\beta y$$

$$y=\alpha A^T x+\beta y$$

$$y=\alpha A^H x+\beta y$$

where α and β are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

on entry

trans character*1
trans specifies the operation to be performed as follows:

TRANS = 'N' or 'n' : matrix is A
TRANS = 'T' or 't' : matrix is A^T
TRANS = 'C' or 'c' : matrix is A^H

m integer
m specifies the number of rows of the matrix A. *m* must be at least zero.

n integer
n specifies the number of columns of the matrix A. *n* must be at least zero.

kl integer
kl specifies the number of sub-diagonals of the matrix *a*. *kl* must satisfy $kl \geq 0$.

ku integer
ku specifies the number of super-diagonals of the matrix *a*. *ku* must satisfy $ku \geq 0$.

alpha Single precision real for sgbmv
Double precision real for dgbmv
Single precision complex for cgbmv
Double precision complex for zgbmv

alpha specifies the scalar α.

a Single precision real for sgbmv
Double precision real for dgbmv
Single precision complex for cgbmv
Double precision complex for zgbmv
array of DIMENSION (LDA, n)
Before entry, the leading *(kl + ku + 1)* by *n* part of the array *A* must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row *(ku + 1)* of the array, the first super-diagonal starting at position 2 in row *ku*, the first sub-diagonal starting at position 1 in row *(ku + 2)*, and so on. Elements in the array *a* that do not correspond to elements in the band matrix (such as the top left *ku* by *ku* triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
      DO 20, J = 1, N
         K = KU + 1 - J
         DO 10, I = MAX( 1, J - KU ), MIN( M, J + KL )
            A( K + I, J ) = matrix( I, J )
10       CONTINUE
20    CONTINUE
```

lda integer
lda specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least *(kl + ku + 1)*.

x Single precision real for sgbmv
Double precision real for dgbmv
Single precision complex for cgbmv
Double precision complex for zgbmv
array of DIMENSION at least

`1 + |incx| * (n - 1)` when TRANS = 'N' or 'n'

and at least

`1 + |incx| * (m - 1)` otherwise.

Before entry, the incremented array *x* must contain the vector *x*.

incx integer
incx specifies the increment for the elements of *x*. *incx* must not be zero.

beta Single precision real for sgbmv
Double precision real for dgbmv
Single precision complex for cgbmv
Double precision complex for zgbmv
beta specifies the scalar β. When *beta* is supplied as zero, *y* need not be set on

input.

y Single precision real for sgbmv
Double precision real for dgbmv
Single precision complex for cgbmv
Double precision complex for zgbmv

array of DIMENSION at least

`1 + |`*incy*`| * (`*m*` - 1)` when TRANS = 'N' or 'n'
and at least
`1 + |`*incy*`| * (`*n*` - 1)` otherwise.

Before entry, the incremented array *y* must contain the vector *y*.

incy integer
incy specifies the increment for the elements of *y*. *incy* must not be zero.

on return

y contains the updated vector *y*.

sgemm/dgemm/cgemm/zgemm

Syntax

call sgemm/dgemm/cgemm/zgemm
(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

Purpose

sgemm/dgemm/cgemm/zgemm performs one of the matrix-matrix operations:

$$C = \alpha AB + \beta C$$

$$C = \alpha A^T B + \beta C$$

$$C = \alpha A^H B + \beta C$$

$$C = \alpha AB^T + \beta C$$

$$C = \alpha A^T B^T + \beta C$$

$$C = \alpha A^H B^T + \beta C$$

$$C = \alpha AB^H + \beta C$$

$$C = \alpha A^T B^H + \beta C$$

$$C = \alpha A^H B^H + \beta C$$

Where α and β are scalars, C is an m by n matrix, A is an n by k or k by n matrix, and B is a k by n or n by k matrix.

On Entry

transa . character*1

specifies the operation to be performed on A:

transa = 'N' or 'n' $C = \alpha AD + \beta C$

transa = 'T' or 't' $C = \alpha A^T D + \beta C$

transa = 'C' or 'c' $C = \alpha A^H D + \beta C$

where D is B, B^T, or B^H.

transb character*1

specifies the operation to be performed on *B*:

transb = 'N' or 'N' $C=\alpha DB+\beta C$

transb = 'T' or 'T' $C=\alpha DB^T+\beta C$

transb = 'C' or 'c' $C=\alpha DB^H+\beta C$

where D is A, A^T, or A^H.

m integer

m specifies the number of rows of the matrix *C*. *m* must be at least zero.

n integer

n specifies the number of columns of the matrix *C*. *n* must be at least zero.

k integer

k specifies the number of columns (rows) of the matrix *A* if *transa* = 'N' ('T', 'C'), and the number of rows (columns) of the matrix B if *transb* = 'N' ('T', 'C'). *k* must be at least zero.

alpha Single precision real for sgemm
Double precision real for dgemm
Single precision complex for cgemm
Double precision complex for zgemm

alpha specifies the scalar α.

a Single precision real for sgemm
Double precision real for dgemm
Single precision complex for cgemm
Double precision complex for zgemm

array of dimension (*lda*, *).
The leading *m* by *k* (*k* by *m*) part of the array *a* must contain the matrix of coefficients for *transa* = 'N' ('T', 'C').

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least *m* (*k*).

b Single precision real for sgemm
Double precision real for dgemm
Single precision complex for cgemm
Double precision complex for zgemm

array of dimension (*ldb*, *).
The leading *k* by *n* (*n* by *k*) part of the array *b* must contain the matrix of coefficients for transb = 'N' ('T', 'C').

ldb integer

ldb specifies the first dimension of *b* as declared in the calling (sub) program. *ldb* must be at least *k* (*n*).

beta Single precision real for sgemm
Double precision real for dgemm
Single precision complex for cgemm
Double precision complex for zgemm

beta specifies the scalar β. When *beta* is supplied as zero then *c* need not be set on input.

c Single precision real for sgemm
Double precision real for dgemm
Single precision complex for cgemm
Double precision complex for zgemm

array of dimension (*ldc*, *n*).
On entry, the leading *m* by *n* part of the array *c* may contain the matrix of coefficients.

ldc integer

specifies the first dimension of *C* as declared in the calling (sub) program. *ldc* must be at least *m*.

On Return

c contains the resultant array *C*.

sgemv/dgemv/cgemv/zgemv

Syntax

call sgemv/dgemv/cgemv/zgemv (trans, m, n, alpha, a, lda, x, incx, beta, y, incy)

Purpose

sgemv/dgemv/cgemv/zgemv performs one of the matrix-vector operations

$$y=\alpha Ax+\beta y$$

$$y=\alpha A^{T}x+\beta y$$

$$y=\alpha A^{H}x+\beta y$$

where α and β are scalars, x and y are vectors and A is an m by n matrix.

On Entry

trans character*1

trans specifies the operation to be performed as follows:

trans = 'N' or 'n' $y=\alpha Ax+\beta y$

trans = 'T' or 't' $y=\alpha A^{T}x+\beta y$

trans = 'C' or 'c' $y=\alpha A^{H}x+\beta y$

m integer

specifies the number of rows of the matrix *a*. *m* must be at least zero.

n integer

specifies the number of columns of the matrix *a*. *n* must be at least zero.

alpha Single precision real for sgemv
Double precision real for dgemv
Single precision complex for cgemv
Double precision complex for zgemv

alpha specifies the scalar α.

a

Single precision real for sgemv
Double precision real for dgemv
Single precision complex for cgemv
Double precision complex for zgemv

array of dimension (*lda*, *n*).
On entry, the leading *m* by *n* part of the array *a* must contain the matrix of coefficients.

lda

integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, m)`.

x

Single precision real for sgemv
Double precision real for dgemv
Single precision complex for cgemv
Double precision complex for zgemv

array of dimension at least
`1 + |incx|*(n - 1)` when *trans* = 'N' or 'n' and at least
`1 + |incx|*(m - 1)` otherwise.
On entry, the incremented array *x* must contain the vector *x*.

incx

integer

specifies the increment for the elements of *x*. *incx* must not be zero.

beta

Single precision real for sgemv
Double precision real for dgemv
Single precision complex for cgemv
Double precision complex for zgemv

beta specifies the scalar β. When *beta* is supplied as zero, *y* need not be set on input.

y

Single precision real for sgemv
Double precision real for dgemv
Single precision complex for cgemv
Double precision complex for zgemv

array of dimension at least

`1 + |incy|*(m - 1)` when *trans* = 'N' or 'n'

and at least

`1 + |incy|*(n - 1)` otherwise.

On entry with *beta* non-zero, the incremented array *y* must contain the vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y is overwritten by the updated vector *y*.

sger/dger

Syntax

call sger/dger (m, n, alpha, x, incx, y, incy, a, lda)

Purpose

sger/dger performs the rank 1 operation

$$A=\alpha x y^T+A$$

where α is a scalar, x is an m-element vector, y is an n-element vector and A is an m by n matrix.

On Entry

m integer

specifies the number of rows of the matrix *a*. *m* must be at least zero.

n integer

specifies the number of columns of the matrix *a*. *n* must be at least zero.

alpha Single precision real for sger
Double precision real for dger

specifies the scalar α.

x Single precision real for sger
Double precision real for dger

array of dimension at least `1+|incx|*(m-1)`.

Contains the *m*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

y Single precision real for sger
Double precision real for dger

array of dimension at least `1+|incy|*(n-1)`.

contains the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

a Single precision real for sger
Double precision real for dger

array of dimension (*lda*, *n*).

On entry, the leading *m* by *n* part of the array *a* must contain the matrix of coefficients.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, m)`.

On Return

a overwritten by the updated matrix.

cgerc/zgerc

Syntax

call cgerc/zgerc (m, n, alpha, x, incx, y, incy, a, lda)

Purpose

cgerc/zgerc performs the rank 1 operation

$$A = \alpha x y^H + A$$

where α is a scalar, x is an m-element vector, y is an n-element vector, and A is an m by n matrix.

On Entry

m integer

specifies the number of rows of the matrix *a*. *m* must be at least zero.

n integer

specifies the number of columns of the matrix *a*. *n* must be at least zero.

alpha Single precision complex for cgerc
Double precision complex for zgerc

specifies the scalar α.

x Single precision complex for cgerc
Double precision complex for zgerc

array of dimension at least `1+|`*incx*`|*(`*m*`-1)`.
Contains the *m*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

y Single precision complex for cgerc
Double precision complex for zgerc

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.
The incremented array *y* contains the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

a Single precision complex for cgerc
Double precision complex for zgerc

array of dimension (*lda*, *n*).
The leading *m* by *n* part of the array *a* must contain the matrix of coefficients.

lda integer

specifies the first dimension of *a*. *lda* must be at least *m*.

On Return

a overwritten by the updated matrix.

cgeru/zgeru

Syntax

call cgeru/zgeru (m, n, alpha, x, incx, y, incy, a, lda)

Purpose

cgeru/zgeru performs the rank 1 operation

$$A = \alpha x y^T + A$$

where α is a scalar, x is an m-element vector, y is an n-element vector and A is an m by n matrix.

On Entry

m integer

specifies the number of rows of the matrix *a*. *m* must be at least zero.

n integer

specifies the number of columns of the matrix *a*. *n* must be at least zero.

alpha Single precision complex for cgeru
Double precision complex for zgeru

specifies the scalar α.

x Single precision complex for cgeru
Double precision complex for zgeru

array of dimension at least `1+|`*incx*`|*(`*m*`-1)`.
Before entry, the incremented array *x* must contain the *m*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

y Single precision complex for cgeru
Double precision complex for zgeru

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.
On entry, the incremented array *y* must contain the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

a Single precision complex for cgeru
Double precision complex for zgeru

array of dimension (*lda*, *n*).
The leading *m* by *n* part of the array *a* must contain the matrix of coefficients.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, m)`.

On Return

a overwritten by the updated matrix.

chbmv/zhbmv

Syntax

call chbmv/zhbmv (uplo, n, k, alpha, a, lda, x, incx, beta, y, incy)

Purpose

chbmv/zhbmv perform the matrix-vector operation

$$y = \alpha A x + \beta y,$$

where α and β are scalars, x and y are n-element vectors and A is an n by n hermitian band matrix, with k super-diagonals.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the band matrix *a* is being supplied as follows:

uplo = 'U' or 'u' : the upper triangular part of *a* is being supplied.

uplo = 'L' or 'l' : the lower triangular part of *a* is being supplied.

n integer

specifies the order of the matrix *a*. *n* must be at least zero.

k integer

specifies the number of super-diagonals of the matrix *a*. *k* must be at least zero.

alpha Single precision complex for chbmv
Double precision complex for zhbmv

specifies the scalar α.

a Single precision complex for chbmv
Double precision complex for zhbmv

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading (*k*+1) by *n* part of the array *a* must contain the upper triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row (*k*+1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. The top left *k* by *k* triangle of the array *a* is not referenced. The following program segment will transfer the upper triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
      do 20 j = 1, n
        m = k + 1 - j
        do 10, i = max(1, j - k), j
          a(m + i, j) = matrix(i, j)
10      continue
20    continue
```

Before entry with *uplo* = 'L' or 'l', the leading (*k*+1) by n part of the array *a* must contain the lower triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right *k* by *k* triangle of the array *a* is not referenced. The following program segment will transfer the lower triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
      do 20 j = 1, n
        m = 1 - j
        do 10 i = j, min(n, j + k)
          a(m + i, j) = matrix(i, j)
10      continue
20    continue
```

Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least (*k*+1).

x Single precision complex for chbmv
Double precision complex for zhbmv

array of dimension at least $1+|incx|*(n-1)$.
Before entry, the incremented array *x* must contain the vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

beta Single precision complex for chbmv
Double precision complex for zhbmv

specifies the scalar β.

y Single precision complex for chbmv
Double precision complex for zhbmv

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.

Incremented array *y* must contain the vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y overwritten by the updated vector *y*.

chemm/zhemm

Syntax

call chemm/zhemm (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)

Purpose

chemm/zhemm perform one of the matrix-matrix operations

$$C = \alpha A B + \beta C,$$

or

$$C = \alpha B A + \beta C,$$

where α and β are scalars, A is a hermitian matrix, and B and C are m by n matrices.

On Entry

side character*1

specifies whether the hermitian matrix A appears on the left or right in the operation as follows:

side = 'L' or 'l' $C = \alpha A B + \beta C,$

side = 'R' or 'r' $C = \alpha B A + \beta C,$

uplo character*1

specifies whether the upper or lower triangular part of the hermitian matrix a is to be referenced as follows:

uplo = 'U' or 'u' only the upper triangular part of the hermitian matrix is to be referenced.

uplo = 'L' or 'l' only the lower triangular part of the hermitian matrix is to be referenced.

m integer

specifies the number of rows of the matrix c. m must be at least zero.

n integer

specifies the number of columns of the matrix c. n must be at least zero.

alpha Single precision complex for chemm
Double precision complex for zhemm

specifies the scalar α.

a

Single precision complex for chemm
Double precision complex for zhemm

array of dimension (*lda*, *ka*), where *ka* is *m* when *side* = 'L' or 'l' and is *n* otherwise.

Before entry with *side* = 'L' or 'l', the *m* by *m* part of the array *a* must contain the hermitian matrix, such that when *uplo* = 'U' or 'u', the leading *m* by *m* upper triangular part of the array *a* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *a* is not referenced, and when *uplo* = 'L' or 'l', the leading *m* by *m* lower triangular part of the array *a* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *a* is not referenced.

Before entry with *side* = 'R' or 'r', the *n* by *n* part of the array *a* must contain the hermitian matrix, such that when *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *a* is not referenced, and when *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *a* is not referenced.

Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

lda

integer

specifies the first dimension of *a* as declared in the calling (sub) program. When *side* = 'L' or 'l' then *lda* must be at least `max(1, m)`, otherwise *lda* must be at least `max(1, n)`.

b

Single precision complex for chemm
Double precision complex for zhemm

array of dimension (*ldb*, *n*).

The leading *m* by *n* part of the array *b* contains the matrix *b*.

ldb

integer

specifies the first dimension of *b* as declared in the calling (sub) program. *ldb* must be at least `max(1, m)`.

beta

Single precision complex for chemm
Double precision complex for zhemm

specifies the scalar β. When *beta* is zero, *c* need not be set on entry.

c

Single precision complex for chemm
Double precision complex for zhemm

array of dimension (*ldc*, *n*).

The leading *m* by *n* part of the array *c* contains the matrix *C*, except when *beta* is zero, in which case *c* need not be set on entry.

ldc integer

specifies the first dimension of *c* as declared in the calling (sub) program. *ldc* must be at least `max(1,m)`.

On Return

c is overwritten by the *m* by *n* updated matrix.

chemv/zhemv

Syntax

call chemv/zhemv (uplo, n, alpha, a, lda, x, incx, beta, y, incy)

Purpose

chemv/zhemv performs the matrix-vector operation

$$y = \alpha A x + \beta y$$

where α and β are scalars, x and y are n-element vectors and A is an n by n hermitian matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *a* is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of *a* is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of *a* is to be referenced.

n integer

specifies the order of the matrix *a*. *n* must be at least zero.

alpha Single precision complex for chemv
Double precision complex for zhemv

specifies the scalar α.

a Single precision complex for chemv
Double precision complex for zhemv

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *a* is not referenced. Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *a* is not referenced. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

x Single precision complex for chemv
Double precision complex for zhemv

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

The incremented array *x* contains the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

beta Single precision complex for chemv
Double precision complex for zhemv

specifies the scalar β. When *beta* is zero, *y* need not be set on input.

y Single precision complex for chemv
Double precision complex for zhemv

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.

The incremented array *y* must contain the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y overwritten by the updated vector *y*.

cher/zher

Syntax

call cher/zher (uplo, n, alpha, x, incx, a, lda)

Purpose

cher/zher performs the hermitian rank 1 operation

$$A = \alpha x x^H + A$$

where α is a real scalar, x is an n-element vector and A is an n by n hermitian matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *a* is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of *a* is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of *a* is to be referenced.

n integer

specifies the order of the matrix *A*. *n* must be at least zero.

alpha Single precision real for cher
Double precision real for zher

specifies the scalar α.

x Single precision complex for cher
Double precision complex for zher

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

Before entry, the incremented array *x* must contain the *n* element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

a Single precision complex for cher
Double precision complex for zher

array of dimension (*lda*, *n*).

On entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *a* is not referenced.
On entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *a* is not referenced.

Note that the imaginary parts of the diagonal elements need not be set; they are assumed to be zero.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

On Return

a With *uplo* = 'U' or 'u', the upper triangular part of the array *a* is overwritten by the upper triangular part of the updated matrix.
With *uplo* = 'L' or 'l', the lower triangular part of the array *a* is overwritten by the lower triangular part of the updated matrix.

The imaginary parts of the diagonal elements are set to zero.

cher2/zher2

Syntax

call cher2/zher2 (uplo, n, alpha, x, incx, y, incy, a, lda)

Purpose

cher2/zher2 performs the hermitian rank 2 operation

$$A = \alpha x y^H + y(\alpha x)^H + A$$

where α is a scalar, x and y are n-element vectors, and A is an n by n hermitian matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *a* is to be referenced as follows:

uplo = 'U' or 'u' only the upper triangular part of *a* is to be referenced.

uplo = 'L' or 'l' only the lower triangular part of *a* is to be referenced.

n integer

specifies the order of the matrix *a*. *n* must be at least zero.

alpha Single precision complex for cher2
Double precision complex for zher2

specifies the scalar α.

x Single precision complex for cher2
Double precision complex for zher2

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

Before entry, the incremented array *x* must contain the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

y Single precision complex for cher2
Double precision complex for zher2

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.

Before entry, the incremented array *y* must contain the *n* element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

a Single precision complex for cher2
Double precision complex for zher2

array of dimension (*lda*, *n*).

On entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *a* is not referenced.

On entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *a* is not referenced.

Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

On Return

a With *uplo* = 'U' or 'u', the upper triangular part of the array *a* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l' the lower triangular part of the array *a* is overwritten by the lower triangular part of the updated matrix.

The imaginary parts of the diagonal elements are set to zero.

cher2k/zher2k

Syntax

call cher2k/zher2k (uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

Purpose

cher2k/zher2k performs one of the hermitian rank 2k operations

$$C = \alpha A B^H + \bar{\alpha} B A^H + \beta C,$$

or

$$C = \alpha A^H B + \bar{\alpha} A B^H + \beta C,$$

where α and β are scalars with β real, C is an n by n hermitian matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array c is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of c is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of c is to be referenced.

trans character*1

specifies the operation to be performed as follows:

trans = 'N' or 'n' :

$$c = \alpha a b^H + \bar{\alpha} b a^H + \beta c.$$

trans = 'C' or 'c' :

$$c = \alpha a^H b + \bar{\alpha} b^H a + \beta c.$$

n integer

specifies the order of the matrix c. n must be at least zero.

k integer

On entry with *trans* = 'N' or 'n', k specifies the number of columns of the matrices a and b, and on entry with *trans* = 'C' or 'c', k specifies the number of rows of the matrices a and b. k must be at least zero.

alpha Single precision complex for cher2k
Double precision complex for zher2k

specifies the scalar α.

a Single precision complex for cher2k
Double precision complex for zher2k

array of dimension (*lda*, *ka*), where *ka* is *k* when *trans* = 'N' or 'n', and is *n* otherwise.

On entry with *trans* = 'N' or 'n', the leading *n* by *k* part of the array *a* must contain the matrix *a*, otherwise the leading *k* by *n* part of the array *a* must contain the matrix *a*.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. When *trans* = 'N' or 'n' then *lda* must be at least max(1, *n*), otherwise *lda* must be at least max(1, *k*).

b Single precision complex for cher2k
Double precision complex for zher2k

array of dimension (*ldb*, *kb*), where *kb* is *k* when *trans* = 'N' or 'n', and is *n* otherwise.

On entry with *trans* = 'N' or 'n', the leading *n* by *k* part of the array *b* must contain the matrix *b*, otherwise the leading *k* by *n* part of the array *b* must contain the matrix *b*.

ldb integer

specifies the first dimension of *b* as declared in the calling (sub) program. When *trans* = 'N' or 'n' then *ldb* must be at least max(1, *n*), otherwise *ldb* must be at least max(1, *k*).

beta Single precision real for cher2k
Double precision real for zher2k

specifies the scalar β.

c Single precision complex for cher2k
Double precision complex for zher2k

array of dimension (*ldc*, *n*).

On entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *c* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *c* is not referenced.

On entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *c* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *c* is not referenced. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

ldc integer

specifies the first dimension of *c* as declared in the calling (sub) program. *ldc* must be at least `max(1, n)`.

On Return

c With *uplo* = 'U' or 'u', the upper triangular part of the array *c* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l' the lower triangular part of the array *c* is overwritten by the lower triangular part of the updated matrix.

The imaginary parts of the diagonal elements are set to zero.

cherk/zherk

Syntax

call cherk/zherk (uplo, transa, n, k, alpha, a, lda, beta, c, ldc)

Purpose

cherk/zherk performs one of the hermitian rank k operations

$$C = \alpha A A^H + \beta C,$$

or

$$C = \alpha A^H A + \beta C,$$

where α and β are real scalars, C is an n by n hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array c is to be referenced as follows:

uplo = 'U' or 'u' only the upper triangular part of c is to be referenced.

uplo = 'L' or 'l' only the lower triangular part of c is to be referenced.

transa character*1

specifies the operation to be performed as follows:

transa = 'N' or 'n' : $c = \alpha a a^H + \beta c$,

transa = 'C' or 'c' : $c = \alpha a^H a + \beta c$.

n integer

specifies the order of the matrix c. n must be at least zero.

k integer

on entry with *transa* = 'N' or 'n', k specifies the number of columns of the matrix a, and on entry with *transa* = 'C' or 'c', k specifies the number of rows of the matrix a. k must be at least zero.

alpha Single precision real for cherk
Double precision real for zherk

specifies the scalar α.

a Single precision complex for cherk
Double precision complex for zherk

array of dimension (*lda*, *ka*), where *ka* is *k* when *transa* = 'N' or 'n', and is *n* otherwise.

On entry with *transa* = 'N' or 'n', the leading *n* by *k* part of the array *a* must contain the matrix *A*, otherwise the leading *k* by *n* part of the array *a* must contain the matrix *A*.

lda integer

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, n)`, otherwise *lda* must be at least `max(1, k)`.

beta Single precision real for cherk
Double precision real for zherk

specifies the scalar β.

c Single precision complex for cherk
Double precision complex for zherk

array of dimension (*ldc*, *n*).

On entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *c* must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of *c* is not referenced.

On entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *c* must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of *c* is not referenced. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

ldc integer

specifies the first dimension of *c* as declared in the calling (sub) program. *ldc* must be at least `max(1, n)`.

On Return

c With *uplo* = 'U' or 'u', the upper triangular part of the array *c* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the lower triangular part of the array *c* is overwritten by the lower triangular part of the updated matrix.

The imaginary parts of the diagonal elements are set to zero.

chpmv/zhpmv

Syntax

call chpmv/zhpmv (uplo, n, alpha, ap, x, incx, beta, y, incy)

Purpose

chpmv/zhpmv performs the matrix-vector operation

$$y = \alpha A x + \beta y,$$

where α and β are scalars, x and y are n-element vectors and A is an n by n hermitian matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array *ap* as follows:

uplo = 'U' or 'u' the upper triangular part of A is supplied in *ap*.

uplo = 'L' or 'l' the lower triangular part of A is supplied in *ap*.

n integer

specifies the order of the matrix A. n must be at least zero.

alpha Single precision complex for chpmv
Double precision complex for zhpmv

specifies the scalar α.

ap Single precision complex for chpmv
Double precision complex for zhpmv

array of dimension at least $(n*(n+1))/2$.

Before entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that *ap*(1) contains $A(1, 1)$, *ap*(2) and *ap*(3) contain $A(1, 2)$ and $A(2, 2)$ respectively, and so on.

Before entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that *ap*(1) contains $A(1, 1)$, *ap*(2) and *ap*(3) contain $A(2, 1)$ and $A(3, 1)$ respectively, and so on.

Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

x Single precision complex for chpmv
Double precision complex for zhpmv

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

Before entry, the incremented array *x* must contain the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

beta Single precision complex for chpmv
Double precision complex for zhpmv

specifies the scalar β. When *beta* is supplied as zero then *y* need not be set on input.

y Single precision complex for chpmv
Double precision complex for zhpmv

array of dimension at least `1+|`*incy*`|*(`*n*`-1)` .

Before entry, the incremented array *y* must contain the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y overwritten by the updated vector *y*.

chpr/zhpr

Syntax

call chpr/zhpr (uplo, n, alpha, x, incx, ap)

Purpose

chpr/zhpr performs the hermitian rank 1 operation

$$A = \alpha x \times x^H + A,$$

where α is a real scalar, x is an n-element vector and A is an n by n hermitian matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array *ap* as follows:

uplo = 'U' or 'u' : the upper triangular part of A is supplied in *ap*.

uplo = 'L' or 'l' : the lower triangular part of A is supplied in *ap*.

n integer

specifies the order of the matrix A. n must be at least zero.

alpha Single precision real for chpr
Double precision real for zhpr

specifies the scalar α.

x Single precision complex for chpr
Double precision complex for zhpr

array of dimension at least (1+|*incx*|*(*n*-1)).
The incremented array x must contain the n-element vector x.

incx integer

specifies the increment for the elements of x. *incx* must not be zero.

ap Single precision complex for chpr
Double precision complex for zhpr

array of dimension at least ((*n*(*n* + 1))/2).

On entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that *ap*(1) contains A(1, 1), *ap*(2) and *ap*(3) contain A(1, 2) and A(2, 2) respectively, and so

on.

On entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(2, 1) and *A*(3, 1) respectively, and so on.

Note that the imaginary parts of the diagonal elements need not be set; they are assumed to be zero.

On Return

ap With *uplo* = 'U' or 'u', the array *ap* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the array *ap* is overwritten by the lower triangular part of the updated matrix.

The imaginary parts of the diagonal elements are set to zero.

chpr2/zhpr2

Syntax

call chpr2/zhpr2 (uplo, n, alpha, x, incx, y, incy, ap)

Purpose

chpr2/zhpr2 performs the rank-1 update:

$$A = \alpha x y^H + \bar{\alpha} y x^H + A,$$

where α is a scalar, x and y are n-element vectors and A is an n by n hermitian matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the matrix a is supplied in the packed array *ap* as follows:

uplo = 'U' or 'u' the upper triangular part of A is supplied in *ap*.

uplo = 'L' or 'l' the lower triangular part of A is supplied in *ap*.

n integer

specifies the order of the matrix A. n must be at least zero.

alpha Single precision complex for chpr2
Double precision complex for zhpr2

specifies the scalar α.

x Single precision complex for chpr2
Double precision complex for zhpr2

array of dimension at least `1+|incx|*(n-1)`.

The incremented array x contains the n-element vector x.

incx integer

specifies the increment for the elements of x. *incx* must not be zero.

y Single precision complex for chpr2
Double precision complex for zhpr2

array of dimension at least `( 1+|incy|*(n-1) )`.

The incremented array y contains the n-element vector y.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

ap Single precision complex for chpr2
Double precision complex for zhpr2

array of dimension at least (*n**(*n*+1))/2.

Before entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(1, 2) and *A*(2, 2) respectively, and so on.

Before entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(2, 1) and *A*(3, 1) respectively, and so on.

Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

On Return

ap With *uplo* = 'U' or 'u', the array *ap* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the array *ap* is overwritten by the lower triangular part of the updated matrix.

The imaginary parts of the diagonal elements are set to zero.

snrm2/dnrm2/scnrm2/dznrm2

Syntax

w = snrm2/dnrm2/scnrm2/dznrm2 (n, x, incx)

Purpose

snrm2/dnrm2/scnrm2/dznrm2 computes the *Euclidean norm* of the elements of a vector:

$$\left[\sum |x_i|^2\right]^{1/2}$$

On Entry

n integer

the order of vector *x*.

x Single precision real for snrm2
Double precision real for dnrm2
Single precision complex for scnrm2
Double precision complex for dznrm2

array of dimension *idimx*
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*.

On Return

w Single precision real for snrm2
Double precision real for dnrm2
Single precision real for scnrm2
Double precision real for dznrm2

norm (square root of the sum of squares) of elements of *x*.

srot/drot

Syntax

call srot/drot (n, dx, incx, dy, incy, c, s)

Purpose

srot/drot apply the Givens transformation matrix *G* (created by **_rotg**) to the *2* by *n* matrix:

$$\begin{bmatrix} x_1 & x_2 & \cdots \\ y_1 & y_2 & \cdots \end{bmatrix}$$

On Entry

n — integer
width of the array.

dx — Single precision real for srot
Double precision real for drot
array of dimension *idimx*.

contains the *x* (upper) components of the input array. The dimension *idimx* must be at least `1+|`*incx*`|*(`*n*`-1)`.

incx — Single precision real for srot
Double precision real for drot

spacing increment for *dx*.

dy — Single precision real for srot
Double precision real for drot
array of dimension *idimy*.

contains the *y* (lower) components of the input array. The dimension *idimy* must be at least `1+|`*incy*`|*(`*n*`-1)`.

incy — Single precision real for srot
Double precision real for drot

spacing increment for *dy*.

c,s Single precision real for srot
Double precision real for drot

c contains the diagonal elements of the transformation matrix *G*. *s* contains the (±) off-diagonal elements of the transformation matrix *G*.

$$G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

On Return

dx holds the *x* (upper) components of the 2 by *n* resulting matrix.

dy holds the *y* (lower) components of the 2 by *n* resulting matrix.

srotg/drotg

Syntax

call srotg/drotg (da, db, dc, ds)

Purpose

srotg/drotg construct the Givens transformation matrix G which zeros the second component of the 2-vector $(da, db)^T$:

$$\begin{bmatrix} da' \\ 0 \end{bmatrix} = G \begin{bmatrix} da \\ db \end{bmatrix}$$

where

$$G = \begin{bmatrix} dc & ds \\ -ds & dc \end{bmatrix}$$

and

$$dc^2 + ds^2 = 1.$$

On Entry

da Single precision real for srotg
Double precision real for drotg

contains the x component of the input vector.

db Single precision real for srotg
Double precision real for drotg

contains the y component of the input vector.

On Return

da Single precision real for srotg
Double precision real for drotg

contains $\pm (da^2 + db^2)^{1/2}$.

db Single precision real for srotg
Double precision real for drotg

contains the value z which allows dc and ds to be recovered by the following algorithm:

$z=1$: $dc=0$ and $ds=1$.

$|z|<1$: $dc=(1-z^2)^{1/2}$ and $ds=z$.

$|z|>1$: $dc=1/z$ and $ds=(1-dc^2)^{1/2}$.

dc Single precision real for srotg
Double precision real for drotg

contains the diagonal components of the resulting matrix.

ds Single precision real for srotg
Double precision real for drotg

contains the off-diagonal components of the resulting matrix.

srotm/drotm

Syntax

call srotm/drotm (n, dx, incx, dy, incy, dparam)

Purpose

srotm/drotm apply the modified Givens transformation matrix *H* (created by **_rotmg**) to the 2 by *n* matrix:

$$\begin{bmatrix} x_1 & x_2 & \cdots \\ y_1 & y_2 & \cdots \end{bmatrix}$$

On Entry

n
integer
width of the array.

dx
Single precision real for srotm
Double precision real for drotm
array of dimension *idimx*.

contains the *x* (upper) components of the input array. The dimension *idimx* must be at least `1+|`*incx*`|*(`*n*`-1)`.

incx
Single precision real for srotm
Double precision real for drotm

spacing increment for *dx*.

dy
Single precision real for srotm
Double precision real for drotm
array of dimension *idimy*.

contains the *y* (lower) components of the input array. The dimension *idimy* must be at least `1+|`*incy*`|*(`*n*`-1)`.

incy
Single precision real for srotm
Double precision real for drotm

spacing increment for *dy*.

dparam Single precision real for srotm
Double precision real for drotm

array of dimension 5.

dparam(1) contains the switch *dflag*.
dparam(2-5) contain *dh11*, *dh21*, *dh12*, and *dh22*, respectively, the components of the array *H*.

The switch *dflag* selects which components of *H* are set in *dparam* and which are implicit:

$$dflag=-1. \; : \; H=\begin{bmatrix} dh11 & dh12 \\ dh21 & dh22 \end{bmatrix}$$

$$dflag=0. \; : \; H=\begin{bmatrix} 1. & dh12 \\ dh21 & 1. \end{bmatrix}$$

$$dflag=1. \; : \; H=\begin{bmatrix} dh11 & 1. \\ -1. & dh22 \end{bmatrix}$$

$$dflag=-2. \; : \; H=\begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

The values 1., -1., and 0. implied by the last three cases are not stored in *dparam(2-5)*.

On Return

dx holds the *x* (upper) components of the 2 by *n* resulting matrix.

dy holds the *y* (lower) components of the 2 by *n* resulting matrix.

srotmg/drotmg

Syntax

call srotmg/drotmg (dd1, dd2, dx1, dy1)

Purpose

srotmg/drotmg construct the modified Givens transformation matrix *H* which zeros the second component of the 2-vector (sqrt*(dd1) dx1*, sqrt*(dd2) dy1*):

$$\begin{bmatrix} x' \\ 0 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

On Entry

dd1
Single precision real for srotmg
Double precision real for drotmg

contains the scaling factor for the x component of the input vector.

dd2
Single precision real for srotmg
Double precision real for drotmg

contains the scaling factor for the y component of the input vector.

dx1
Single precision real for srotmg
Double precision real for drotmg

contains the x component of the input vector.

dy1
Single precision real for srotmg
Double precision real for drotmg

contains the y component of the input vector.

On Return

dparam
Single precision real for srotmg
Double precision real for drotmg

array of dimension 5.

dparam(1) contains the switch *dflag*.
dparam(2-5) contain *dh11*, *dh21*, *dh12*, and *dh22*, respectively, the components of the array *H*.

The switch *dflag* selects which components of *H* are set in *dparam* and which are implicit.

$$dflag=-1. \; : \; H = \begin{bmatrix} dh11 & dh12 \\ dh21 & dh22 \end{bmatrix}$$

$$dflag=0. \; : \; H = \begin{bmatrix} 1. & dh12 \\ dh21 & 1. \end{bmatrix}$$

$$dflag=1. \; : \; H = \begin{bmatrix} dh11 & 1. \\ -1. & dh22 \end{bmatrix}$$

$$dflag=-2. \; : \; H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

The values 1., -1., and 0. implied by the last three cases are not stored in *dparam(2-5)*.

ssbmv/dsbmv

Syntax

call ssbmv/dsbmv (uplo, n, k, alpha, a, lda, x, incx, beta, y, incy)

Purpose

ssbmv/dsbmv performs the matrix-vector operation

$$y = \alpha A x + \beta y,$$

where α and β are scalars, x and y are n-element vectors and A is an n by n symmetric band matrix, with k super-diagonals.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the band matrix *a* is being supplied as follows:

uplo = 'U' or 'u' the upper triangular part of *A* is being supplied.

uplo = 'L' or 'l' the lower triangular part of *A* is being supplied.

n integer

specifies the order of the matrix *A*. *n* must be at least zero.

k integer

specifies the number of super-diagonals of the matrix *A*. *k* must satisfy $k \geq 0$.

alpha Single precision real for ssbmv
Double precision real for dsbmv

specifies the scalar α.

a Single precision real for ssbmv
Double precision real for dsbmv

array of dimension (*lda*, *n*).
On entry with uplo = 'U' or 'u', the leading (*k* + 1) by *n* part of the array *a* must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (*k* + 1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. The top left *k* by *k* triangle of the array *a* is not referenced. The following program segment will transfer the upper triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
      do 20, j = 1, n
        m = k + 1 - j
        do 10, i = max( 1, j - k ), j
           a( m + i, j ) = matrix( i, j )
10      continue
20    continue
```

On entry with *uplo* = 'L' or 'l', the leading ($k+1$) by n part of the array a must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array a is not referenced. The following program segment will transfer the lower triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
      do 20, j = 1, n
        m = 1 - j
        do 10, i = j, min( n, j + k )
          a( m + i, j ) = matrix( i, j )
10      continue
20    continue
```

lda integer

specifies the first dimension of a as declared in the calling (sub) program. *lda* must be at least ($k + 1$).

x Single precision real for ssbmv
Double precision real for dsbmv

array of dimension at least ($1+|incx|*(n-1)$).
On entry, the array x contains the vector x.

incx integer

specifies the increment for the elements of x. *incx* must not be zero.

beta Single precision real for ssbmv
Double precision real for dsbmv

specifies the scalar β.

y Single precision real for ssbmv
Double precision real for dsbmv

array of dimension at least ($1+|incy|*(n-1)$).
On entry, the array y contains the vector y.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y overwritten by the updated vector *y*.

sscal/dscal/cscal/zscal/csscal/zdscal

Syntax

call sscal/dscal/cscal/zscal/csscal/zdscal (n, alpha, x, incx)

Purpose

sscal/dscal/cscal/zscal/csscal/zdscal multiplies a vector by a scalar.

$$x = \alpha x$$

where α is a scalar and x is an n-element vector. **cscal/zscal** multiplies a complex vector by a complex number. **csscal/zsscal** multiplies a complex vector by a real scalar.

On Entry

n integer

the order of vector *x*.

alpha Single precision real for sscal and csscal
Double precision real for dscal and zdscal
Single precision complex for cscal
Double precision complex for zscal

the scalar α.

x Single precision real for sscal
Double precision real for dscal
Single precision complex for cscal and csscal
Double precision complex for zscal and zdscal

array of dimension *idimx*
contains the spaced vector *x* of order *n*. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector *x*.

On Return

x contains the resultant spaced vector *x* of order *n*.

sspmv/dspmv

Syntax

call sspmv/dspmv (uplo, n, alpha, ap, x, incx, beta, y, incy)

Purpose

sspmv/dspmv performs the matrix-vector operation

$$y = \alpha A x + \beta y$$

where α and β are scalars, x and y are n-element vectors and A is an n by n symmetric matrix, supplied in packed form.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array *ap* as follows:

uplo = 'U' or 'u' : the upper triangular part of A is supplied in *ap*.

uplo = 'L' or 'l' : the lower triangular part of A is supplied in *ap*.

n integer

specifies the order of the matrix A. *n* must be at least zero.

alpha Single precision real for sspmv
Double precision real for dspmv

specifies the scalar α.

ap Single precision real for sspmv
Double precision real for dspmv

array of dimension at least (*n**(*n*+1))/2.

On entry with uplo = 'U' or 'u', the array *ap* must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that *ap*(1) contains $A(1, 1)$, *ap*(2) and *ap*(3) contain $A(1, 2)$ and $A(2, 2)$ respectively, and so on.

On entry with uplo = 'L' or 'l', the array *ap* must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that *ap*(1) contains $A(1, 1)$, *ap*(2) and *ap*(3) contain $A(2, 1)$ and $A(3, 1)$ respectively, and so on.

x Single precision real for sspmv
Double precision real for dspmv

array of dimension at least (1+|*incx*|*(*n*-1)).
The incremented array *x* must contain the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

beta Single precision real for sspmv
Double precision real for dspmv

specifies the scalar β. When *beta* is supplied as zero, *y* need not be set on input.

y Single precision real for sspmv
Double precision real for dspmv

array of dimension at least 1+|*incy*|*(*n*-1).

The incremented array *y* must contain the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y overwritten by the updated vector *y*.

sspr/dspr

Syntax

call sspr/dspr (uplo, n, alpha, x, incx, ap)

Purpose

sspr/dspr performs the symmetric rank 1 operation

$$A = \alpha x x^T A$$

where α is a real scalar, x is an n-element vector and A is an n by n symmetric matrix, supplied in packed form.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array *ap* as follows:

uplo = 'U' or 'u' the upper triangular part of A is supplied in *ap*.

uplo = 'L' or 'l' the lower triangular part of A is supplied in *ap*.

n integer

specifies the order of the matrix A. n must be at least zero.

alpha Single precision real for sspr
Double precision real for dspr

specifies the scalar α.

x Single precision real for sspr
Double precision real for dspr

array of dimension at least `1+|incx|*(n-1)`.

On entry, the incremented array x contains the n-element vector x.

incx integer

specifies the increment for the elements of x. *incx* must not be zero.

ap Single precision real for sspr
Double precision real for dspr

array of dimension at least `n*(n+1)/2`.

On entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular part

of the symmetric matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(1, 2) and *A*(2, 2), respectively, and so on.

Before entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(2, 1) and *A*(3, 1) respectively, and so on.

On Return

ap With *uplo* = 'U' or 'u', *ap* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the array *ap* is overwritten by the lower triangular part of the updated matrix.

sspr2/dspr2

Syntax

call sspr2/dspr2 (uplo, n, alpha, x, incx, y, incy, ap)

Purpose

sspr2/dspr2 performs the symmetric rank 2 operation

$$A = \alpha x y^T + \alpha y x^T + A$$

where α is a scalar, x and y are n-element vectors and A is an n by n symmetric matrix, supplied in packed form.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array *ap* as follows:

uplo = 'U' or 'u' : the upper triangular part of A is supplied in *ap*.

uplo = 'L' or 'l' : the lower triangular part of A is supplied in *ap*.

n integer

specifies the order of the matrix A. n must be at least zero.

alpha Single precision real for sspr2
Double precision real for dspr2

specifies the scalar α.

x Single precision real for sspr2
Double precision real for dspr2

array of dimension at least `1+|incx|*(n-1)`.

On entry, the incremented array x contains the n-element vector x.

incx integer

specifies the increment for the elements of x. *incx* must not be zero.

y Single precision real for sspr2
Double precision real for dspr2

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.

On entry, the incremented array *y* contains the n-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

ap Single precision real for sspr2
Double precision real for dspr2

array of dimension at least *n*`*(`*n*`+1)/2`.

On entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(1, 2) and *A*(2, 2) respectively, and so on.

Before entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(2, 1) and *A*(3, 1) respectively, and so on.

On Return

ap With *uplo* = 'U' or 'u', the array *ap* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the array *ap* is overwritten by the lower triangular part of the updated matrix.

sswap/dswap/cswap/zswap

Syntax

call sswap/dswap/cswap/zswap (n, x, incx, y, incy)

Purpose

sswap/dswap/cswap/zswap exchanges (swaps) a vector with another vector:

$$y=x_{old}$$

$$x=y_{old}$$

where x and y are n-element vectors.

On Entry

n integer

the order of vectors x and y.

x Single precision real for sswap
Double precision real for dswap
Single precision complex for cswap
Double precision complex for zswap

array of dimension *idimx*
contains the spaced vector x of order n. The dimension *idimx* must be at least as large as `1+|incx|*(n-1)`.

incx integer

spacing increment for vector x.

y Single precision real for sswap
Double precision real for dswap
Single precision complex for cswap
Double precision complex for zswap

array of dimension *idimy*
contains the spaced vector y of order n. The dimension *idimy* must be at least as large as `1+|incx|*(n-1)`.

incy integer

spacing increment for vector y.

On Return

x	contains the resultant vector *x*.
y	contains the resultant vector *y*.

ssymm/dsymm/csymm/zsymm

Syntax

call ssymm/dsymm/csymm/zsymm (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)

Purpose

ssymm/dsymm/csymm/zsymm performs one of the matrix-matrix operations

$$C = \alpha A B + \beta C,$$

or

$$C = \alpha B A + \beta C,$$

where α and β are scalars, A is a symmetric matrix and B and C are m by n matrices.

On Entry

side character*1

specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

side = 'L' or 'l' : $C=\alpha AB+\beta C$

side = 'R' or 'r' : $C=\alpha BA+\beta C$

uplo character*1

specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of the symmetric matrix is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of the symmetric matrix is to be referenced.

m integer

specifies the number of rows of the matrix C. m must be at least zero.

n integer

specifies the number of columns of the matrix C. n must be at least zero.

alpha

Single precision real for ssymm
Double precision real for dsymm
Single precision complex for csymm
Double precision complex for zsymm

specifies the scalar α.

a

Single precision real for ssymm
Double precision real for dsymm
Single precision complex for csymm
Double precision complex for zsymm

array of dimension (*lda, ka*), where *ka* is *m* when *side* = 'L' or 'l' and is *n* otherwise.

On entry with *side* = 'L' or 'l', the *m* by *m* part of the array *A* must contain the symmetric matrix, such that when *uplo* = 'U' or 'u', the leading *m* by *m* upper triangular part of the array *A* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *A* is not referenced, and when *uplo* = 'L' or 'l', the leading *m* by *m* lower triangular part of the array *A* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *A* is not referenced.

Before entry with *side* = 'R' or 'r', the *n* by *n* part of the array *A* must contain the symmetric matrix, such that when *uplo* = 'U' or 'u', the leading n by n upper triangular part of the array *A* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *A* is not referenced, and when *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *A* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *A* is not referenced.

lda

integer

specifies the first dimension of a as declared in the calling (sub) program. when *side* = 'L' or 'l' then *lda* must be at least `max(1, m)`, otherwise *lda* must be at least `max(1, n)`.

b

Single precision real for ssymm
Double precision real for dsymm
Single precision complex for csymm
Double precision complex for zsymm

array of dimension (*ldb, n*).

On entry, the leading *m* by *n* part of the array *b* must contain the matrix *B*.

ldb

integer

specifies the first dimension of *b* as declared in the calling (sub) program. *ldb* must be at least `max( 1, m )`.

beta Single precision real for ssymm
Double precision real for dsymm
Single precision complex for csymm
Double precision complex for zsymm

specifies the scalar β. When *beta* is supplied as zero, *c* need not be set on input.

c Single precision real for ssymm
Double precision real for dsymm
Single precision complex for csymm
Double precision complex for zsymm

array of dimension (*ldc*, *n*).

On 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.

ldc integer

specifies the first dimension of *c* as declared in the calling (sub) program. *ldc* must be at least `max( 1, m )`.

On Return

c overwritten by the *m* by *n* updated matrix.

ssymv/dsymv

Syntax

call ssymv/dsymv (uplo, n, alpha, a, lda, x, incx, beta, y, incy)

Purpose

ssymv/dsymv performs the matrix-vector operation

$$y = \alpha A x + \beta y$$

where α and β are scalars, x and y are n-element vectors and A is an n by n symmetric matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *A* is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of *A* is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of *A* is to be referenced.

n integer

specifies the order of the matrix *A*. *n* must be at least zero.

alpha Single precision real for ssymv
Double precision real for dsymv

specifies the scalar α.

a Single precision real for ssymv
Double precision real for dsymv

array of dimension (*lda*, *n*).
Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* contains the upper triangular part of the symmetric matrix and the strictly lower triangular part of *a* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* contains the lower triangular part of the symmetric matrix and the strictly upper triangular part of *a* is not referenced.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

x Single precision real for ssymv
Double precision real for dsymv

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

The incremented array *x* contains the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

beta Single precision real for ssymv
Double precision real for dsymv

specifies the scalar β. When beta is zero, *y* need not be set on input.

y Single precision real for ssymv
Double precision real for dsymv

array of dimension at least `1+|`*incy*`|*(`*n*`-1)`.

The incremented array *y* contains the *n*-element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

On Return

y is overwritten by the updated vector *y*.

ssyr/dsyr

Syntax

call ssyr/dsyr (uplo, n, alpha, x, incx, a, lda)

Purpose

ssyr/dsyr performs the symmetric rank 1 operation

$$A = \alpha x x^T + A$$

where α is a real scalar, x is an n-element vector and A is an n by n symmetric matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *a* is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of *a* is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of *a* is to be referenced.

n integer

specifies the order of the matrix *a*. *n* must be at least zero.

alpha Single precision real for ssyr
Double precision real for dsyr

specifies the scalar α.

x Single precision real for ssyr
Double precision real for dsyr

array of dimension at least `1+|incx|*(n-1)`.

Before entry, the incremented array *x* contains the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

a Single precision real for ssyr
Double precision real for dsyr

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *a* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *a* is not referenced.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

On Return

a With *uplo* = 'U' or 'u', the upper triangular part of the array *a* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the lower triangular part of the array *a* is overwritten by the lower triangular part of the updated matrix.

ssyr2/dsyr2

Syntax

call ssyr2/dsyr2 (uplo, n, alpha, x, incx, y, incy, a, lda)

Purpose

ssyr2/dsyr2 performs the symmetric rank 2 operation

$$A = \alpha x y^T + \alpha y x^T + A$$

where α is a scalar, x and y are n element vectors and A is an n by n symmetric matrix.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *a* is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of *a* is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of *a* is to be referenced.

n integer

specifies the order of the matrix *a*. *n* must be at least zero.

alpha Single precision real for ssyr2
Double precision real for dsyr2

specifies the scalar α.

x Single precision real for ssyr2
Double precision real for dsyr2

array of dimension at least `1+|incx|*(n-1)`.

Before entry, the incremented array *x* contains the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

y Single precision real for ssyr2
Double precision real for dsyr2

array of dimension at least `1+|incy|*(n-1)`.

Before entry, the incremented array *y* must contain the *n* element vector *y*.

incy integer

specifies the increment for the elements of *y*. *incy* must not be zero.

a Single precision real for ssyr2
Double precision real for dsyr2

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *a* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *a* is not referenced.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

On Return

a With *uplo* = 'U' or 'u', the upper triangular part of the array *a* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the lower triangular part of the array *a* is overwritten by the lower triangular part of the updated matrix.

ssyr2k/dsyr2k/csyr2k/zsyr2k

Syntax

call ssyr2k/dsyr2k/csyr2k/zsyr2k (uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

Purpose

ssyr2k/dsyr2k/csyr2k/zsyr2k performs one of the symmetric rank 2k operations

$$C = \alpha AB^T + \alpha BA^T + \beta C$$

or

$$C = \alpha A^T B + \alpha B^T A + \beta C$$

where α and β are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of C is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of C is to be referenced.

trans character*1

specifies the operation to be performed as follows:

trans = 'N' or 'n' : $C = \alpha A B^T + \alpha B A^T + \beta C$.

trans = 'T' or 't' : $C = \alpha A^T B + \alpha B^T A + \beta C$.

n integer

specifies the order of the matrix *c*. *n* must be at least zero.

k integer

on entry with *trans* = 'N' or 'n', *k* specifies the number of columns of the matrices *a* and *b*, and on entry with *trans* = 'T' or 't' or 'C' or 'c', *k* specifies the number of rows of the matrices *a* and *b*. *k* must be at least zero.

alpha Single precision real for ssyr2k
Double precision real for dsyr2k
Single precision complex for csyr2k
Double precision complex for zsyr2k

specifies the scalar α.

a

Single precision real for ssyr2k
Double precision real for dsyr2k
Single precision complex for csyr2k
Double precision complex for zsyr2k

array of dimension (*lda, ka*), where *ka* is *k* when *trans* = 'N' or 'n', and is *n* otherwise.

Before entry with *trans* = 'N' or 'n', the leading *n* by *k* part of the array *a* must contain the matrix *a*, otherwise the leading *k* by *n* part of the array *a* must contain the matrix *a*.

lda

integer

specifies the first dimension of *a* as declared in the calling (sub) program. When *trans* = 'N' or 'n' then *lda* must be at least max(1, *n*), otherwise *lda* must be at least max(1, *k*).

b

Single precision real for ssyr2k
Double precision real for dsyr2k
Single precision complex for csyr2k
Double precision complex for zsyr2k

array of dimension (*ldb, kb*), where *kb* is *k* when *trans* = 'N' or 'n', and is *n* otherwise.

Before entry with *trans* = 'N' or 'n', the leading *n* by *k* part of the array *b* must contain the matrix *b*, otherwise the leading *k* by *n* part of the array *b* must contain the matrix *b*.

ldb

integer

specifies the first dimension of *b* as declared in the calling (sub) program. When *trans* = 'N' or 'n' then *ldb* must be at least `max(1, n)`, otherwise *ldb* must be at least max(1, *k*).

beta

Single precision real for ssyr2k
Double precision real for dsyr2k
Single precision complex for csyr2k
Double precision complex for zsyr2k

specifies the scalar β.

c

Single precision real for ssyr2k
Double precision real for dsyr2k
Single precision complex for csyr2k
Double precision complex for zsyr2k

array of dimension (*ldc*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *c* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *c* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *c* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *c* is not referenced.

ldc integer

specifies the first dimension of *c* as declared in the calling (sub) program. *ldc* must be at least max(1, *n*).

On Return

c With *uplo* = 'U' or 'u', the upper triangular part of the array *c* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the lower triangular part of the array *c* is overwritten by the lower triangular part of the updated matrix.

ssyrk/dsyrk/csyrk/zsyrk

Syntax

call ssyrk/dsyrk/csyrk/zsyrk (uplo, transa, m, k, alpha, a, lda, beta, c, ldc)

Purpose

ssyrk/dsyrk/csyrk/zsyrk perform one of the symmetric rank *k* operations

$$C = \alpha A A^T + \beta C$$

or

$$C = \alpha A^T A + \beta C$$

where α and β are scalars, C is an m by m symmetric matrix and A is an m by k matrix in the first case and a k by m matrix in the second case.

On Entry

uplo character*1

specifies whether the upper or lower triangular part of the array *c* is to be referenced as follows:

uplo = 'U' or 'u' : only the upper triangular part of C is to be referenced.

uplo = 'L' or 'l' : only the lower triangular part of C is to be referenced.

transa character*1

specifies the operation to be performed as follows:

transa= 'N' or 'n' : $C = \alpha A A^T + \beta C$.

transa= 'T' or 't' : $C = \alpha A^T A + \beta C$.

m integer

specifies the order of the matrix *c*. *m* must be at least zero.

k integer

on entry with *transa*= 'N' or 'n', *k* specifies the number of columns of the matrix *a*, and on entry with *transa*= 'T' or 't' or 'C' or 'c', *k* specifies the number of rows of the matrix *a*. *k* must be at least zero.

alpha Single precision real for ssyrk
Double precision real for dsyrk
Single precision complex for csyrk
Double precision complex for zsyrk

specifies the scalar α.

a Single precision real for ssyrk
Double precision real for dsyrk
Single precision complex for csyrk
Double precision complex for zsyrk

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 contain the matrix *a*, otherwise the leading *k* by *m* part of the array *a* must contain the matrix *A*.

lda integer

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)`.

beta Single precision real for ssyrk
Double precision real for dsyrk
Single precision complex for csyrk
Double precision complex for zsyrk

specifies the scalar β.

c Single precision real for ssyrk
Double precision real for dsyrk
Single precision complex for csyrk
Double precision complex for zsyrk

array of dimension (*ldc*, *m*).

Before entry with *uplo* = 'U' or 'u', the leading *m* by *m* upper triangular part of the array *c* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *c* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *m* by *m* lower triangular part of the array *c* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *c* is not referenced.

ldc integer

specifies the first dimension of *c* as declared in the calling (sub) program. *ldc* must be at least `max(1, m)`.

On Return

c With *uplo* = 'U' or 'u', the upper triangular part of the array *c* is overwritten by the upper triangular part of the updated matrix.

With *uplo* = 'L' or 'l', the lower triangular part of the array *c* is overwritten by the lower triangular part of the updated matrix.

stbmv/dtbmv/ctbmv/ztbmv

Syntax

call stbmv/dtbmv/ctbmv/ztbmv (uplo, trans, diag, n, k, a, lda, x, incx)

Purpose

stbmv/dtbmv/ctbmv/ztbmv perform one of the matrix-vector operations

$$x=Ax$$

$$x=A^{T}x$$

$$x=A^{H}x$$

where x is an n-element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with $(k+1)$ diagonals.

On Entry

uplo character*1

specifies whether the matrix is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' : *a* is an upper triangular matrix.

uplo = 'L' or 'l' : *a* is a lower triangular matrix.

trans character*1

specifies the operation to be performed as follows:

trans = 'N' or 'n' : $x=Ax$.

trans = 'T' or 't' : $x=A^{T}x$.

trans = 'C' or 'c' : $x=A^{H}x$.

diag character*1

specifies whether or not *a* unit triangular as follows:

diag = 'U' or 'u' *a* is assumed to be unit triangular.

diag = 'N' or 'n' *a* is not assumed to be unit triangular.

n integer

specifies the order of the matrix *a*. *n* must be at least zero.

k integer

With *uplo* = 'U' or 'u', *k* specifies the number of super-diagonals of the matrix *A*.

With *uplo* = 'L' or 'l', *k* specifies the number of sub-diagonals of the matrix *A*.

k must be zero or positive.

a Single precision real for stbmv
Double precision real for dtbmv
Single precision complex for ctbmv
Double precision complex for ztbmv

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading (*k*+1) by *n* part of the array *a* must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (*k*+1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. the top left *k* by *k* triangle of the array *a* not referenced.

The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
      do 20, j = 1, n
         m = k + 1 - j
         do 10, i = max(1, j - k), j
            a(m + i, j) = matrix(i, j)
10       continue
20    continue
```

Before entry with *uplo* = 'L' or 'l', the leading (*k*+1) by *n* part of the array *a* must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right *k* by *k* triangle of the array *a* is not referenced. The following program segment will transfer *a* lower triangular band matrix from conventional full matrix storage to band storage:

```
      do 20, j = 1, n
        m = 1 - j
        do 10, i = j, min(n, j + k)
          a(m + i, j) = matrix(i, j)
10      continue
20    continue
```

Note that when *diag* = 'U' or 'u' the elements of the array *a* corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least (*k*+1).

x Single precision real for stbmv
Double precision real for dtbmv
Single precision complex for ctbmv
Double precision complex for ztbmv

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

Before entry, the incremented array *x* contains the *n*-element vector *x*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

On Return

x is overwritten with the transformed vector *x*.

stbsv/dtbsv/ctbsv/ztbsv

Syntax

call stbsv/dtbsv/ctbsv/ztbsv (uplo, trans, diag, n, k, a, lda, x, incx)

Purpose

stbsv/dtbsv/ctbsv/ztbsv solves one of the systems of equations

$$A x = b$$

$$A^T x = b$$

$$A^H x = b$$

where b and x are n-element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with $(k+1)$ diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

On Entry

uplo character*1

specifies whether the matrix is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' A is an upper triangular matrix.

uplo = 'L' or 'l' A is a lower triangular matrix.

trans character*1

specifies the equations to be solved as follows:

trans = 'N' or 'n' : $A x = b$.

trans = 'T' or 't' : $A^T x = b$.

trans = 'C' or 'c' : $A^H x = b$.

diag character*1

specifies whether or not A is unit triangular as follows:

diag = 'U' or 'u' : A is assumed to be unit triangular.

diag = 'N' or 'n' : A is not assumed to be unit triangular.

n integer

specifies the order of the matrix *A*. *n* must be at least zero.

k integer

On entry with *uplo* = 'U' or 'u', *k* specifies the number of super-diagonals of the matrix *a*.

On entry with *uplo* = 'L' or 'l', *k* specifies the number of sub-diagonals of the matrix *a*.
k must satisfy $k \geq 0$.

a Single precision real for stbsv
Double precision real for dtbsv
Single precision complex for ctbsv
Double precision complex for ztbsv

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading (*k*+1) by *n* part of the array *a* must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (*k*+1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. the top left *k* by *k* triangle of the array *a* is not referenced.

The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
      do 20 j = 1, n
         m = k + 1 - j
         do 10, i = max(1, j - k), j
           a(m + i, j) = matrix(i, j)
10       continue
20    continue
```

On entry with *uplo* = 'L' or 'l', the leading (*k*+1) by *n* part of the array *a* must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. the bottom right *k* by *k* triangle of the array *a* is not referenced.

The following program segment will transfer *a* lower triangular band matrix from conventional full matrix storage to band storage:

```
      do 20 j = 1, n
        m = 1 - j
        do 10, i = j, min(n, j + k)
          a(m + i, j) = matrix(i, j)
10      continue
20    continue
```

Note that when *diag* = 'U' or 'u' the elements of the array *a* corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least (*k*+1).

x Single precision real for stbsv
Double precision real for dtbsv
Single precision complex for ctbsv
Double precision complex for ztbsv

array of dimension at least `1+|`*incx*`|*(`*n*`-1)`.

Before entry, the incremented array *x* must contain the *n* element right-hand side vector *b*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

On Return

x overwritten with the solution vector *x*.

stpmv/dtpmv/ctpmv/ztpmv

Syntax

call stpmv/dtpmv/ctpmv/ztpmv (uplo, trans, diag, n, ap, x, incx)

Purpose

stpmv/dtpmv/ctpmv/ztpmv performs one of the matrix-vector operations

$$x = A\,x$$

$$x = A^T x$$

$$x = A^H x$$

where x is an n-element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

On Entry

uplo character*1

specifies whether the matrix is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' : A is an upper triangular matrix.

uplo = 'L' or 'l' : A is a lower triangular matrix.

trans character*1

specifies the operation to be performed as follows:

trans = 'N' or 'n' : $x = A\,x$

trans = 'T' or 't' : $x = A^T x$

trans = 'C' or 'c' : $x = A^H x$ (ctpmv and ztpmv only).

diag character*1

specifies whether or not A is unit triangular as follows:

diag = 'U' or 'u' A is assumed to be unit triangular.

diag = 'N' or 'n' A is not assumed to be unit triangular.

n integer

specifies the order of the matrix A. n must be at least zero.

ap

Single precision real for stpmv
Double precision real for dtpmv
Single precision complex for ctpmv
Double precision complex for ztpmv

array of dimension at least `(n*(n+1))/2`.

Before entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(1, 2) and *A*(2, 2), respectively, and so on.

Before entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(2, 1) and *A*(3, 1), respectively, and so on.

Note that when *diag* = 'U' or 'u', the diagonal elements of *A* are not referenced, but are assumed to be unity.

x

Single precision real for stpmv
Double precision real for dtpmv
Single precision complex for ctpmv
Double precision complex for ztpmv

array of dimension at least `1+|incx|*(n-1)`.

Before entry, the incremented array *x* contains the *n*-element vector *x*.

incx

integer

specifies the increment for the elements of *x*. *incx* must not be zero.

On Return

x

is overwritten with the transformed vector *x*.

stpsv/dtpsv/ctpsv/ztpsv

Syntax

call stpsv/dtpsv/ctpsv/ztpsv (uplo, trans, diag, n, ap, x, incx)

Purpose

stpsv/dtpsv/ctpsv/ztpsv solves one of the systems of equations

$$A\,x = b$$

$$A^T x = b$$

$$A^H x = b$$

where b and x are n-element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

On Entry

uplo character*1

specifies whether the matrix is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' A is an upper triangular matrix.

uplo = 'L' or 'l' A is a lower triangular matrix.

trans character*1

specifies the equations to be solved as follows:

trans = 'N' or 'n' $A\,x = b$

trans = 'T' or 't' $A^T x = b$

trans = 'C' or 'c' $A^H x = b$

diag character*1

specifies whether or not A is unit triangular as follows:

diag = 'U' or 'u' A is assumed to be unit triangular.

diag = 'N' or 'n' A is not assumed to be unit triangular.

n integer

specifies the order of the matrix *A*. *n* must be at least zero.

ap Single precision real for stpsv
Double precision real for dtpsv
Single precision complex for ctpsv
Double precision complex for ztpsv

array of dimension at least *n** (*n*+1) /2.

Before entry with *uplo* = 'U' or 'u', the array *ap* must contain the upper triangular matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(1, 2) and *A*(2, 2) respectively, and so on.

Before entry with *uplo* = 'L' or 'l', the array *ap* must contain the lower triangular matrix packed sequentially, column by column, so that *ap*(1) contains *A*(1, 1), *ap*(2) and *ap*(3) contain *A*(2, 1) and *A*(3, 1) respectively, and so on.

Note that when *diag* = 'U' or 'u', the diagonal elements of *A* are not referenced, but are assumed to be unity.

x Single precision real for stpsv
Double precision real for dtpsv
Single precision complex for ctpsv
Double precision complex for ztpsv

array of dimension at least 1+|*incx*|*(*n*-1).

Before entry, the incremented array *x* contains the *n*-element right-hand side vector *b*.

incx integer

specifies the increment for the elements of *x*. *incx* must not be zero.

On Return

x overwritten with the solution vector *x*.

strmm/dtrmm/ctrmm/ztrmm

Syntax

call strmm/dtrmm/ctrmm/ztrmm (side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

Purpose

strmm/dtrmm/ctrmm/ztrmm perform one of the matrix-matrix operations

$$B = \alpha\, op(A)\, B$$

$$B = \alpha B\, op(A)$$

where α is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and $op(A)$ is one of:

$$op(A) = A,$$

$$op(A) = A^T,$$

$$op(A) = A^H.$$

On Entry

side character*1

specifies whether $op(A)$ multiplies B from the left or right as follows:

side = 'L' or 'l' : $B = \alpha\, op(A)\, B$.

side = 'R' or 'r' : $B = \alpha B\, op(A)$.

uplo character*1

specifies whether the matrix *a* is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' : A is an upper triangular matrix.

uplo = 'L' or 'l' : A is a lower triangular matrix.

transa character*1

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' : $op(A) = A^T$

transa = 'C' or 'c' : $op(A) = A^H$

diag character*1

specifies whether or not *A* is unit triangular as follows:

diag = 'U' or 'u' : *A* is assumed to be unit triangular.

diag = 'N' or 'n' : *A* is not assumed to be unit triangular.

m integer

specifies the number of rows of *B*. *m* must be at least zero.

n integer

specifies the number of columns of *B*. *n* must be at least zero.

alpha Single precision real for strmm
Double precision real for dtrmm
Single precision complex for ctrmm
Double precision complex for ztrmm

specifies the scalar α.
When *alpha* is zero, *a* is not referenced and *b* need not be set before entry.

a Single precision real for strmm
Double precision real for dtrmm
Single precision complex for ctrmm
Double precision complex for ztrmm

array of dimension (*lda*, *k*), where *k* is *m* when *side* = 'L' or 'l' and is *n* when *side* = 'R' or 'r'.

Before entry with *uplo* = 'U' or 'u', the leading *k* by *k* upper triangular part of the array *a* must contain the upper triangular matrix and the strictly lower triangular part of *A* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *k* by *k* lower triangular part of the array *a* must contain the lower triangular matrix and the strictly upper triangular part of *A* is not referenced.

Note that when *diag* = 'U' or 'u', the diagonal elements of *A* are not referenced either, but are assumed to be unity.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program.

When *side* = 'L' or 'l' then *lda* must be at least `max(1, m)`.

When *side* = 'R' or 'r' then *lda* must be at least `max(1, n)`.

b Single precision real for strmm
Double precision real for dtrmm
Single precision complex for ctrmm
Double precision complex for ztrmm

array of dimension (*ldb*, *n*).

Before entry, the leading *m* by *n* part of the array *b* must contain the matrix *B*.

ldb integer

specifies the first dimension of *b* as declared in the calling (sub) program. *ldb* must be at least `max(1, m)`.

On Return

b is overwritten by the transformed matrix.

strmv/dtrmv/ctrmv/ztrmv

Syntax

call strmv/dtrmv/ctrmv/ztrmv (uplo, trans, diag, n, a, lda, x, incx)

Purpose

strmv/dtrmv/ctrmv/ztrmv performs one of the matrix-vector operations

$$x = A\,x$$

$$x = A^T x$$

$$x = A^H x$$

where x is an n-element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

On Entry

uplo character*1

specifies whether the matrix is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' : A is an upper triangular matrix.

uplo = 'L' or 'l' : A is a lower triangular matrix.

trans character*1

specifies the operation to be performed as follows:

trans = 'N' or 'n' : $x = A\,x$.

trans = 'T' or 't' : $x = A^T x$.

trans = 'C' or 'c' : $x = A^H x$.

diag character*1

specifies whether or not A is unit triangular as follows:

diag = 'U' or 'u' : A is assumed to be unit triangular.

diag = 'N' or 'n' : A is not assumed to be unit triangular.

n integer

specifies the order of the matrix a. n must be at least zero.

a

Single precision real for strmv
Double precision real for dtrmv
Single precision complex for ctrmv
Double precision complex for ztrmv

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular matrix and the strictly lower triangular part of *a* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular matrix and the strictly upper triangular part of *a* is not referenced.

Note that when *diag* = 'U' or 'u', the diagonal elements of *a* are not referenced either, but are assumed to be unity.

lda

integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

x

Single precision real for strmv
Double precision real for dtrmv
Single precision complex for ctrmv
Double precision complex for ztrmv

array of dimension at least `1+|incx|*(n-1)`.

Before entry, the incremented array *x* must contain the *n*-element vector *x*.

incx

integer

specifies the increment for the elements of *x*. *incx* must not be zero.

On Return

x

overwritten with the transformed vector *x*.

strsm/dtrsm/ctrsm/ztrsm

Syntax

call strsm/dtrsm/ctrsm/ztrsm (side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

Purpose

strsm/dtrsm/ctrsm/ztrsm solves one of the matrix equations

$$op(A)X=\alpha B$$

or

$$X\,op(A)=\alpha B$$

where α is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and $op(A)$ is one of

$$op(A)=A,$$

$$op(A)=A^T,$$

$$op(A)=A^H.$$

The matrix X is overwritten on B.

On Entry

side character*1

specifies whether $op(a)$ appears on the left or right of X as follows:

side = 'L' or 'l' $op(A)X=\alpha B$

side = 'R' or 'r' $X\,op(A)=\alpha B$

uplo character*1

specifies whether the matrix A is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' : A is an upper triangular matrix.

uplo = 'L' or 'l' : A is a lower triangular matrix.

transa character*1

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' : $op(A)=A^T$

transa = 'C' or 'c' : $op(A)=A^H$

diag character*1

specifies whether or not *A* is unit triangular as follows:

diag = 'U' or 'u' : *A* is assumed to be unit triangular.

diag = 'N' or 'n' : *A* is not assumed to be unit triangular.

m integer

specifies the number of rows of *B*. *m* must be at least zero.

n integer

specifies the number of columns of *B*. *n* must be at least zero.

alpha Single precision real for strsm
Double precision real for dtrsm
Single precision complex for ctrsm
Double precision complex for ztrsm

specifies the scalar α.
When *alpha* is zero then *a* is not referenced and *b* need not be set before entry.

a Single precision real for strsm
Double precision real for dtrsm
Single precision complex for ctrsm
Double precision complex for ztrsm

array of dimension (*lda*, *k*), where *k* is *m* when *side* = 'L' or 'l' and *n* when *side* = 'R' or 'r'.

Before entry with *uplo* = 'U' or 'u', the leading *k* by *k* upper triangular part of the array *a* must contain the upper triangular matrix and the strictly lower triangular part of *a* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *k* by *k* lower triangular part of the array *a* must contain the lower triangular matrix and the strictly upper triangular part of *a* is not referenced.

Note that when *diag* = 'U' or 'u', the diagonal elements of *a* are not referenced either, but are assumed to be unity.

lda integer

specifies the first dimension of *a* as declared in the calling (sub) program. When *side* = 'L' or 'l' then *lda* must be at least `max(1, m)`. When *side* = 'R' or 'r' then *lda* must be at least `max(1, n)`.

b Single precision real for strsm
Double precision real for dtrsm
Single precision complex for ctrsm
Double precision complex for ztrsm

array of dimension (*ldb*, *n*).

Before entry, the leading *m* by *n* part of the array *b* must contain the right-hand side matrix *B*.

ldb integer

specifies the first dimension of *b* as declared in the calling (sub) program. *ldb* must be at least `max(1, m)`.

On Return

b is overwritten by the solution matrix *X*.

strsv/dtrsv/ctrsv/ztrsv

Syntax

call strsv/dtrsv/ctrsv/ztrsv (uplo, trans, diag, n, a, lda, x, incx)

Purpose

strsv/dtrsv/ctrsv/ztrsv solves one of the systems of equations

$$A\,x = b$$

$$A^T x = b$$

$$A^H x = b$$

where b and x are n-element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

On Entry

uplo character*1

specifies whether the matrix is an upper or lower triangular matrix as follows:

uplo = 'U' or 'u' : A is an upper triangular matrix.

uplo = 'L' or 'l' : A is a lower triangular matrix.

trans character*1

specifies the equations to be solved as follows:

trans = 'N' or 'n' : $A\,x = b$

trans = 'T' or 't' : $A^T x = b$

trans = 'C' or 'c' : $A^H x = b$

diag character*1

specifies whether or not A is unit triangular as follows:

diag = 'U' or 'u' : A is assumed to be unit triangular.

diag = 'N' or 'n' : A is not assumed to be unit triangular.

n integer

specifies the order of the matrix A. n must be at least zero.

a

Single precision real for strsv
Double precision real for dtrsv
Single precision complex for ctrsv
Double precision complex for ztrsv

array of dimension (*lda*, *n*).

Before entry with *uplo* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular matrix and the strictly lower triangular part of *a* is not referenced.

Before entry with *uplo* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *a* must contain the lower triangular matrix and the strictly upper triangular part of *a* is not referenced.

Note that when *diag* = 'U' or 'u', the diagonal elements of *a* are not referenced either, but are assumed to be unity.

lda

integer

specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least `max(1, n)`.

x

Single precision real for strsv
Double precision real for dtrsv
Single precision complex for ctrsv
Double precision complex for ztrsv

array of dimension at least `1+|incx|*(n-1)`.

Before entry, the incremented array *x* must contain the *n*-element right-hand side vector *b*.

incx

integer

specifies the increment for the elements of *x*. *incx* must not be zero.

On Return

x

overwritten with the solution vector *x*.

2.2. Fourier Transform Routines

The following sections describe the *Fast Fourier Transform* routines included in the *Basic Math Library*. The routines are in alphabetical order.

cfft1d/zfft1d *fast fourier transform (in place)*

Syntax

call cfft1d/zfft1d (r, n, isign, wsave)

Purpose

These routines compute either the forward fft or inverse fft of a complex vector (single or double precision).

if (*isign* = -1) compute the forward fft

if (*isign* = 1) compute the inverse fft

if (*isign* = 0) initialize fft coefficients for both the forward and inverse fft.

A call to **cfft1d/zfft1d** with *isign* = 0 must be made prior to using that routine to compute either the forward or inverse fft for a given length in order to initialize coefficients. Thereafter, any number of transforms of the same length can be computed using these coefficients.

On Entry

r

single precision complex for cfft1d
double precision complex for zfft1d

array of dimension at least (*n*)

contains the vector on which the transform is to be performed. *r* is not referenced if *isign* = 0.

n

integer

transform length. *n* must be a power of 2.

isign

integer

flag indicating the type of operation to be performed.
isign = 0 initialize the coefficients *wsave*
= -1 perform the forward fft
= 1 perform the inverse fft.

wsave

single precision complex for cfft1d
double precision complex for zfft1d

array of dimension at least (*n*+2)

If *isign* = 1 or -1 then it contains the fft coefficients initialized on a previous call with *isign* = 0. If *isign* = 0 then it does not need to be defined on input.

On Return

r	If *isign* = 1 or *isign* = -1, it contains the result of the transform. It is not changed if *isign* = 0.
wsave	If *isign* = 0 it contains the initialized fft coefficients. It is unaltered otherwise.

csfft1d/zdfft1d *1d complex-to-real inverse fft*

Syntax

call csfft1d/zdfft1d (r, n, isign, wsave)

Purpose

These routines perform the 1d inverse fft of a complex vector which represents the forward transform of real sequence. The operation is done in place.

A call to either csfft1d or scfft1d (zdfft1d or dzfft1d) with *isign* = 0 must be made prior to computing either the forward or inverse fft for a given length in order to initialize coefficients. Thereafter, any number of transforms of the same length can be computed using these coefficients.

The results of csfft1d/zdfft1d are properly scaled.

On Entry

r

single precision real for csfft1d
double precision real for zdfft1d

array of dimension at least (*n*+2)

If *isign* = 0, it is not referenced.
If *isign* ≠ 0, it represents the complex transform of a real sequence in the following sense: Let z be the complex sequence of length *n* defined by

```
z(i) = cmplx(r(2*i+1),r(2*i+2)),     i = 0 to n/2,
z(n/2+i) = conjg(z(n/2-i)),          i = 0 to n/2 -1,
```

with *r*(2) taken to be 0. Then *z*(0:*n*-1) is the forward fft of some real sequence of length *n*.

n

integer

transform length. It must be a power of 2.

isign

integer

flag indicating the operation to be performed:
isign = 0, initialize the coefficients *wsave*.
≠ 0, perform the inverse fft.

wsave

single precision real for csfft1d
double precision real for zdfft1d

array of dimension at least (2**n* + 4).

If *isign* = 0, *wsave* does not need to be defined on entry. Otherwise, *wsave* contains coefficients required to perform the fft that have been initialized on a previous call to either csfft1d or scfft1d (zdfft1d or dzfft1d) with *isign* = 0 using the same value of *n*.

On Return

r	If *isign* = 0, it is not altered. If *isign* ≠ 0, *r*(1:*n*) contains the real sequence whose transformation is *z*, as defined for **On Entry**, above.
wsave	If *isign* = 0 it contains the coefficients required by scfft1d and csfft1d (zdfft1d and dzfft1d). If *isign* ≠ 0 it is unaltered.

scfft1d/dzfft1d *1d real-to-complex forward fft*

Syntax

call scfft1d/dzfft1d (r, n, isign, wsave)

Purpose

These routines perform the 1d forward fft on a real (single or double precision) vector in place.

A call to either csfft1d or scfft1d (zdfft1d or dzfft1d) with *isign* = 0 must be made prior to computing either the forward or inverse fft for a given length in order to initialize coefficients. Thereafter, any number of transforms of the same length can be computed using these coefficients.

The results of scfft1d/dzfft1d are properly scaled.

On Entry

r

single precision real for scfft1d
double precision real for dzfft1d

array of dimension at least (*n*+2)

contains the real input sequence to be transformed in *r*(1:*n*). *r*(*n*+1) and *r*(*n*+2) are used on output. *r* is not referenced if *isign* = 0.

n

integer

transform length. It must be a power of 2.

isign

integer

flag indicating the operation to be performed:

isign = 0,	initialize the coefficients in *wsave*.
isign ≠ 0,	perform the forward fft.

wsave

single precision real for scfft1d
double precision real for dzfft1d

array of dimension at least (2**n* + 4)

If *isign* = 0, *wsave* need not be defined. Otherwise *wsave* contains coefficients required to perform the fft that have been initialized on a previous call to either csfft1d or scfft1d (zdfft1d or dzfft1d) with *isign* = 0 using the same value of *n*.

On Return

r If *isign* = 0, it is not altered. If *isign* ≠ 0, it contains the output representing the complex transform of the input sequence in the following sense: Let *z* be the complex sequence of length *n* defined by

```
z(i) = cmplx(r(2*i+1),r(2*i+2)),        i = 0 to n/2,
z(n/2+i) = conjg(z(n/2-i)),             i = 1 to n/2 - 1.
```

Then z(0:n-1) is the forward fft of the input sequence r(1:n).

wsave If *isign* = 0 it contains the coefficients required by scfft1d and csfft1d (zdfft1d and dzfft1d). If *isign* ≠ 0 it is unaltered.

Appendix A

Routine Arguments

A.1. Vector Arguments

Vector arguments are passed in one-dimensional arrays. Associated with the vector are a length and an increment which are passed as integer variables. The length specifies the number of elements in the vector. The increment (also called stride) of the vector specifies the spacing between vector elements and the order of the elements in the one-dimensional array in which the vector is passed. If a vector of length n and increment $incx$ is passed in a one-dimensional array x, then its values stored at $x(1), x(1+|incx|), ..., x(1+(n-1)*|incx|)$. If $incx$ is positive, then the elements are stored in increasing order in the array x. If $incx$ is negative, then the elements are stored in decreasing order with the first element being stored in $x(1+(n-1)*|incx|)$. If $incx$ is zero, then all elements of the vector have the same value which is stored in $x(1)$. The dimension of the one-dimensional array holding the vector must always be at least

$$idimx = 1 + (n-1)*|incx|$$

Example 1: Let $x(1:10)$ be the one-dimensional real array

$$x = (1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0) .$$

If

$$incx = 2 \text{ and } n = 4 ,$$

then the vector argument with elements in order from first to last is

$$(1.0, 3.0, 5.0, 7.0) .$$

If

$$incx = -2 \text{ and } n = 4 ,$$

then the vector argument with elements in order from first to last is

$$(7.0, 5.0, 3.0, 1.0) .$$

If

$$incx = 0 \text{ and } n = 4 ,$$

then the vector argument with elements in order from first to last is

$$(1.0, 1.0, 1.0, 1.0) .$$

One-dimensional substructures of a matrix, such as the rows, columns, and diagonals, can be passed as vector arguments provided the correct starting address and increment are specified. With Fortran column-major ordering used for storing the m-by-n matrix, the increment between elements in the same column is 1, the increment between elements in the same row is m, and the increment between elements on the same diagonal is $m+1$.

Example 2: Let A be the real 5x4 matrix declared as

```
REAL A(5,4).
```

The third column of A can be scaled by 2.0 by invoking the BLAS routine *sscal* with the following statement:

```
CALL SSCAL (5, 2.0, A(1,3), 1).
```

The second row can be scaled by 2.0 with the statement:

```
CALL SSCAL (4, 2.0, A(2,1), 5).
```

The main diagonal of A can be scaled by 2.0 with the statement:

```
CALL SSCAL (5, 2.0, A(1,1), 6).
```

Notes:

(1) Some routines have the restriction that the vector increment cannot be zero. This is to maintain compatibility with the Argonne BLAS interface.

(2) If the increment of a vector argument is not specified, then it is assumed to be 1.

A.2. Matrix Arguments

Matrix arguments are passed in two-dimensional arrays. Fortran column-major ordering of the storage is assumed, i.e., elements of the same column occupy successive storage locations. Associated with a matrix argument are three quantities: its leading dimension which specifies the number of storage locations between elements in the same row, its number of rows, and its number of columns. The leading dimension of the matrix must always be at least as large as the number of rows. In addition, a character transposition parameter is often passed which indicates whether the matrix argument is to be used in normal or transposed form or, if the matrix is complex, if the conjugate transpose of the matrix is to be used. The values of the transposition parameter for these cases are *'N'*, *'T'* and *'C'*, respectively.

Example 3: Suppose $A(1{:}5, 1{:}4)$ is the complex two-dimensional array given by

$$\begin{bmatrix} (1.1,.11) & (1.2,.12) & (1.3,.13) & (1.4,.14) \\ (2.1,.21) & (2.2,.22) & (2.3,.23) & (2.4,.24) \\ (3.1,.31) & (3.2,.32) & (3.3,.33) & (3.4,.34) \\ (4.1,.41) & (4.2,.42) & (4.3,.43) & (4.4,.44) \\ (5.1,.51) & (5.2,.52) & (5.3,.53) & (5.4,.54) \end{bmatrix}.$$

Let *transa* be the transposition parameter, *m* be the number of rows, *n* be the number of columns, and *lda* be the leading dimension. Then if

$$transa = 'N',\ m = 4,\ n = 2,\ lda = 5,$$

the matrix argument would be

$$\begin{bmatrix} (1.1,.11) & (1.2,.12) \\ (2.1,.21) & (2.2,.22) \\ (3.1,.31) & (3.2,.32) \\ (4.1,.41) & (4.2,.42) \end{bmatrix}.$$

If

$$transa = 'T',\ m = 4,\ n = 2,\ lda = 5,$$

the matrix argument would be

$$\begin{bmatrix} (1.1,.11) & (2.1,.21) & (3.1,.31) & (4.1,.41) \\ (1.2,.12) & (2.2,.22) & (3.2,.32) & (4.2,.42) \end{bmatrix}.$$

If

$$transa = 'C',\ m = 4,\ n = 2,\ lda = 5,$$

the matrix argument would be

$$\begin{bmatrix} (1.1,-.11) & (2.1,-.21) & (3.1,-.31) & (4.1,-.41) \\ (1.2,-.12) & (2.2,-.22) & (3.2,-.32) & (4.2,-.42) \end{bmatrix}.$$

Note: Care should be taken when using a leading dimension value which is different from the number of rows specified in the declaration of the two-dimensional array. For example, suppose the array A above is declared as

```
COMPLEX  A(5,4).
```

Then if

$$transa = 'N',\ m = 3,\ n = 4,\ lda = 4,$$

the matrix argument will be

$$\begin{bmatrix} (1.1,.11) & (5.1,.51) & (4.2,.42) & (3.3,.33) \\ (2.1,.21) & (1.2,.12) & (5.2,.52) & (4.3,.43) \\ (3.1,.31) & (2.2,.22) & (1.3,.13) & (5.3,.53) \end{bmatrix}.$$

Appendix B

Examples

This section presents examples illustrating the calling sequence of programs in the Basic Math Library.

Example 1:

The following program illustrates a call to the BLAS 1 routine **saxpy**.

```
      integer n, incx, incy
      real x(5), y(9), alpha
      data n / 5 /, incx / 1 /, incy / -2 /, alpha / 1.1 /
      data x / 0.1, 0.2, 0.3, 0.4, 0.5 /
      data y / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 /
c
      call saxpy( n, alpha, x, incx, y, incy )
c
      write(*,1) y
c
 1      format( 9f8.2 )
      end
```

Output:

```
1.55  2.00  3.44  4.00  5.33  6.00  7.22  8.00  9.11
```

Example 2:

The following program illustrates a call to the BLAS 2 routine sger.

```
      integer m, n, incx, incy, i, j
      real x(3), y(3), alpha, a(4,2)
      data m /3/, n / 2 /, lda /4/
      data incx / 1 /, incy / 2 /, alpha / 0.1 /
      data x / 0.1, 0.2, 0.3/,  y / 1.0, 2.0, 3.0/
      data a / 1.1, 2.1, 3.1, 4.1, 1.2, 2.2, 3.2, 4.2/
c
      call sger( m, n, alpha, x, incx, y, incy, a, lda )
c
      do 1 i = 1, lda
         write(*,2) ( a(i,j), j = 1, n )
 1      continue
c
 2      format( 2f8.2 )
      end
```

Output:

```
1.11  1.23
2.12  2.26
3.13  3.29
4.10  4.20
```

Example 3:

The following program illustrates a call to the BLAS 3 routine **strmm**.

```
      integer m, n, incx, incy, i, j
      real alpha, a(4,3), b(5,2)
      character side, uplo, transa, diag
      data m /3/, n / 2 /, lda /4/, ldb /5/
      data alpha / 1.0 /
      data side / 'L'/, uplo /'U'/, transa /'N'/, diag /'U'/

      data a / 1.1, 2.1, 3.1, 4.1, 1.2, 2.2, 3.2, 4.2, 1.3, 2.3, 3.3, 4.3/

      data b / 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0/
c
      call strmm( side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb )
c
      do 1 i = 1, ldb
         write(*,2) ( b(i,j), j = 1, n )
 1    continue
c
 2    format( 2f8.2 )
      end
```

Output:

```
0.73  2.48
0.89  2.54
0.30  0.80
0.40  0.90
0.50  1.00
```

Example 4:

The following program illustrates calls to the FFT routines **scfft1d** and **csfft1d**:

```
      integer n, iflag
      real r(10)
      complex wsave(10)
      data n /8/
      data r /1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0/

      iflag = 0
      call scfft1d( r, n, iflag , wsave )
      iflag = 1
      call scfft1d( r, n, iflag, wsave )
      write(*,1) r
      iflag = 1
      call csfft1d( r, n, iflag, wsave )
      write(*,1) r
1     format(10f8.2)
      end
```

Output:

```
 36.00  0.00 -4.00  9.66 -4.00  4.00 -4.00  1.66 -4.00  0.00
  1.00  2.00  3.00  4.00  5.00  6.00  7.00  8.00  0.00  0.00
```

Appendix C

Performance Hints

Performance Hints for Intel i860 Processors

This appendix presents suggestions on how to obtain optimal performance from the routines in the Basic Math Library on computers which use the Intel i860 Processor.

Whenever possible, follow these guidelines concerning array arguments to routines in the library:

(1) Vector arguments should have an increment equal to one.

(2) Double precision complex arrays should always begin on a 16-byte boundary, i.e., the memory address of the first element of the array should be divisible by 16. Some Fortran compilers provide an option that will force this alignment. As an alternative, common blocks begin on 16-byte boundaries and so the first array at the beginning of a common block will always have the desired alignment. Subsequent arrays in the common block will also be on a 16-byte boundary.

```
DOUBLE PRECISION COMPLEX X, Y
COMMON X(20), Y(10)
```

will force `X` and `Y` onto 16-byte storage boundaries.

(3) Array arguments to the FFT programs should lie on 16-byte storage boundaries.

(4) Vector arguments to BLAS 1 routines should be in the data cache. If there are two vector arguments to a BLAS 1 routine, it is best if only the first argument be in cache rather than only the second argument. A vector will usually be in cache after it has just been referenced, provided it is small enough (no larger than 8192 bytes). The first loop below will generally perform better than the second, since `X` will be in cache after the first call and is accessed as the first operand while `Y` will not remain in cache.

```
      DO 10 J = 1, 100
        S = S + SDOT( 1000, X, 1, Y(1,J), 1 )
10    CONTINUE

      DO 10 J = 1, 100
         S = S + SDOT( 1000, Y(1,J), 1, X, 1 )
10    CONTINUE
```

There is not much impact if the vector arguments to the BLAS 2 or FFT routines are not in cache because these routines are designed to manage the cache. The BLAS 3 routines also manage the cache but have no vector arguments.

Appendix D

Performance Evaluation of the BLAS Math Library

INTRODUCTION

The BLAS Math Library includes four groups of routines: BLAS1 vector-vector operations, BLAS2 matrix-vector operations, BLAS3 matrix-matrix operations, and FFT routines. The more frequently used routines in this library are coded in i860 Assembler language.

The following sections include discussions of the tests and performance for each group of routines. The main goal of this evaluation is to understand the iPSC/860 single-node performance of non-pathological library usage. A secondary goal is to resolve any problems that were found as part of the acceptance process. These results are strictly for reference only.

BLAS1 TEST CASES AND PERFORMANCE

Most of the test cases for BLAS1 routines were adopted from Intel's VecLib test suites. The test suites for the DSDOT, SDSDOT, CSSCAL, and ZDSCAL routines were added. There are no test suites for the ICAMAX, IZAMAX, SCNRM2, DZNRM2, SROTM, DROTM, SROTMG, and DROTMG routines.

The SASUM, SAXPY, SDOT, SDSDOT, SROT, SSCAL, DASUM, DAXPY, DDOT, DROT, DSCAL, DSDOT, CAXPY, CSCAL, CSSCAL, CDOTC, CDOTU, DZASUM, SCASUM, ZAXPY, ZDOTC, ZDOTU, ZDSCAL, ZSCAL routines were tested for performance and validity. The ISAMAX, SCOPY, SNRM2, SROTG, SSWAP, IDAMAX, DCOPY, DNRM2, DROTG, DSWAP, CCOPY, CSWAP, ZCOPY, ZSWAP routines were validated.

Only the unit stride case was used for evaluation. Maximum N is set for 1750 for all the routines that are tested for performance. All test cases have a built-in self-checking mechanisms. The expected results are calculated in Fortran code and compared with the actual results that are returned from the routine calls.

Performance was taken by calculating Mflops for N with values from 100 to 1700 in increments of 100. For most routines, the value of X is in the cache after the first call. The SASUM, SAXPY, SDOT, SDSDOT, DASUM, DAXPY, DDOT, DSDOT, CAXPY, DZASUM, SCASUM, ZAXPY routines were called 50,000 times for each stepping. The DSCAL SSCAL, ZDSCAL and ZSCAL routines were called 1000 times to avoid floating point overflow or underflow. The DROT routine was called 50 times, CSCAL and CSSCAL routines were called 10 times and the SROT routine was called 5 times.

While testing SROTG, CDOTC, CDOTU, and ZDOTU routines, some of the actual results and expected results vary after the 7th or 8th significant digit. Table D-1 shows the performance achieved to date for single-precision routines. Table D-2 shows the performance achieved to date for double-precision routines. Table D-3 shows the performance achieved to date for complex*8 routines and Table D-4 shows the performance achieved to date for complex*16 routines.

Table D-1. BLAS1 Single-Precision Routines Performance

N	SASUM	SAXPY	SDOT	SDSDOT	SROT	SSCAL
100	4.25	12.55	26.59	5.44	8.98	10.61
200	4.54	15.53	31.09	5.76	8.03	11.37
300	4.64	16.07	34.37	5.87	9.51	11.64
400	4.69	15.98	35.57	5.82	8.49	11.78
500	4.73	15.82	36.67	5.89	9.52	11.87
600	4.75	15.87	36.57	5.92	9.51	11.92
700	4.76	16.31	37.05	5.93	9.50	11.81
800	4.77	15.81	37.53	5.95	9.51	12.00
900	4.78	15.97	37.87	5.96	9.50	11.89
1000	4.77	15.65	38.08	5.95	9.53	11.93
1100	4.76	15.67	38.51	5.78	9.40	11.95
1200	4.76	15.63	39.47	5.62	9.07	11.96
1300	4.76	15.60	40.35	5.48	9.01	11.91
1400	4.75	15.74	41.03	5.40	9.17	11.99
1500	4.75	15.53	41.70	5.32	8.94	11.93
1600	4.75	15.51	42.23	5.26	8.93	11.93
1700	4.75	15.69	42.61	5.20	9.04	11.86

Table D-2. BLAS1 Double-Precision Routines Performance

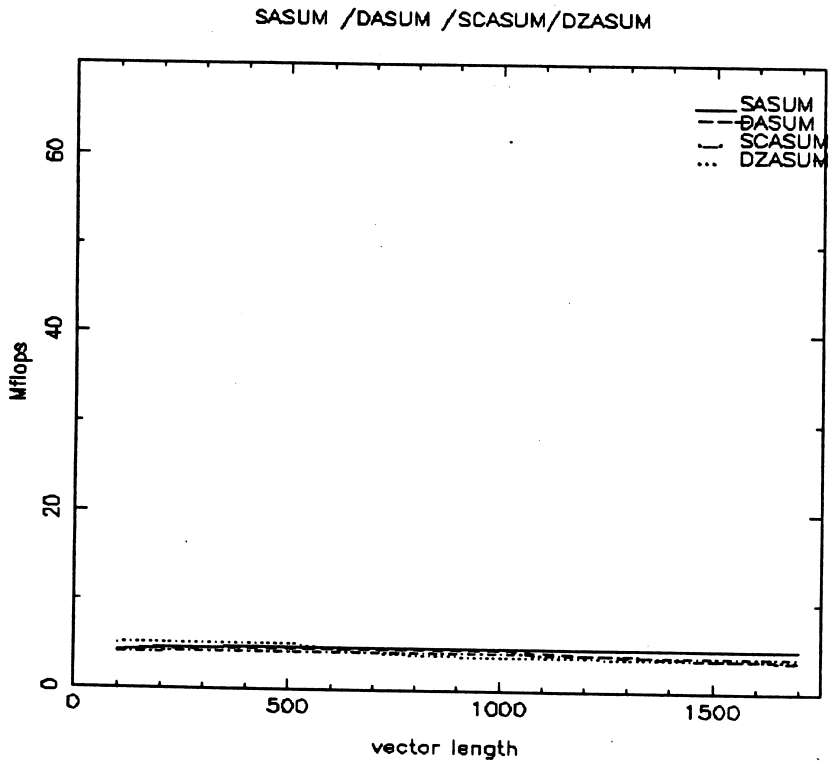
N	DASUM	DAXPY	DDOT	DSDOT	DROT	DSCAL
100	4.18	10.62	17.19	5.46	15.64	10.04
200	4.45	11.09	19.42	5.77	15.83	11.05
300	4.54	11.40	21.35	5.88	16.43	11.41
400	4.59	11.56	21.60	5.83	16.23	11.57
500	4.61	11.66	21.83	5.88	15.86	11.71
600	4.62	11.73	22.77	5.92	14.06	11.78
700	4.64	11.77	24.02	5.94	12.80	11.83
800	4.65	11.81	24.12	5.95	12.15	11.87
900	4.66	11.80	25.33	5.96	11.85	11.86
1000	4.64	11.67	26.44	5.95	11.58	11.87
1100	4.34	10.45	21.69	5.78	11.46	9.86
1200	4.10	9.34	17.66	5.62	11.36	8.37
1300	3.92	8.56	15.26	5.48	11.29	7.42
1400	3.77	8.00	13.71	5.40	11.22	3.23
1500	3.66	7.59	12.58	5.32	11.19	3.23
1600	3.60	7.33	11.93	5.26	11.18	3.23
1700	3.55	7.17	11.56	5.20	11.14	3.23

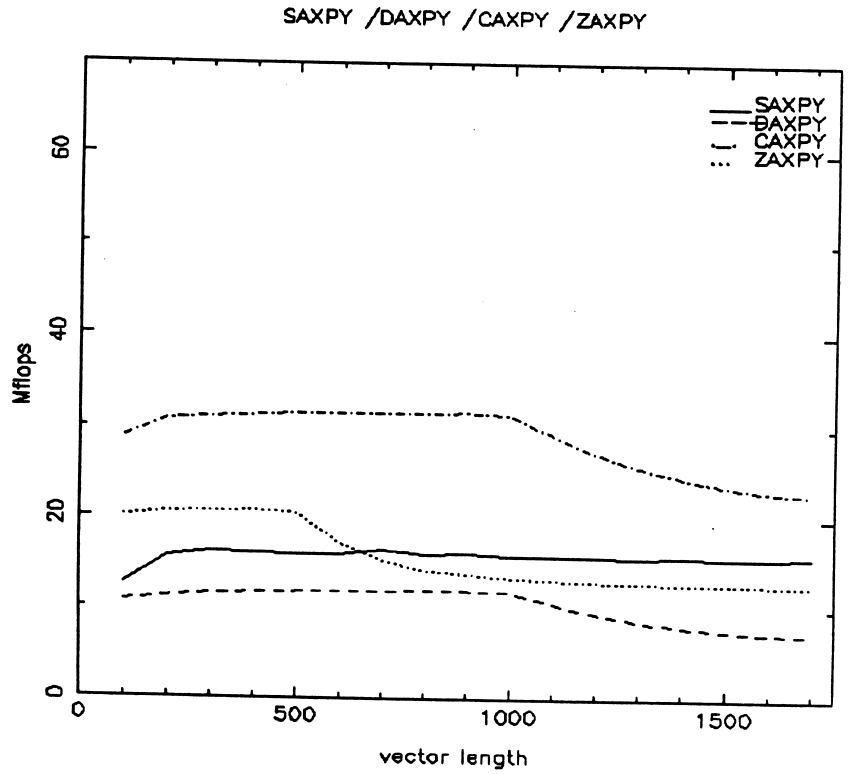
Table D-3. BLAS1 Complex*8 Performance Comparison

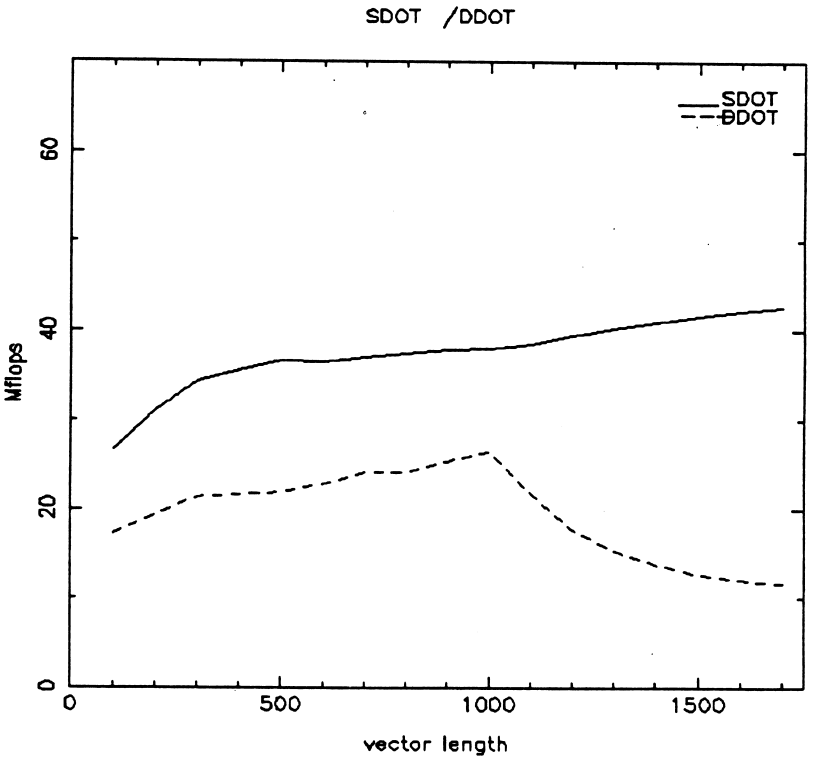
N	SCASUM	CAXPY	CSCAL	CSSCAL	CDOTC	CDOTU
100	4.04	28.77	12.97	27.51	48.72	48.79
200	4.12	30.77	13.39	29.99	52.98	53.13
300	4.15	31.05	13.54	30.74	54.73	54.94
400	4.16	31.22	13.62	31.19	57.14	57.25
500	4.17	31.43	13.65	31.38	58.75	58.77
600	4.15	31.38	13.64	31.35	61.66	61.82
700	4.15	31.48	13.61	31.44	65.02	64.71
800	4.15	31.44	13.66	31.35	68.04	68.81
900	4.16	31.52	13.21	31.22	72.57	
1000	4.16	31.31	13.21	30.36	74.51	
1100	4.11	29.17	13.19	29.27	63.68	
1200	4.05	27.14	12.93	27.66	55.23	
1300	4.00	25.66	12.44	26.50	49.64	
1400	3.95	24.50	12.26	24.73	45.61	
1500	3.92	23.61	12.21	24.96	42.77	
1600	3.90	23.01	12.01	23.75	41.23	
1700	3.88	22.65	11.95	24.23	40.28	

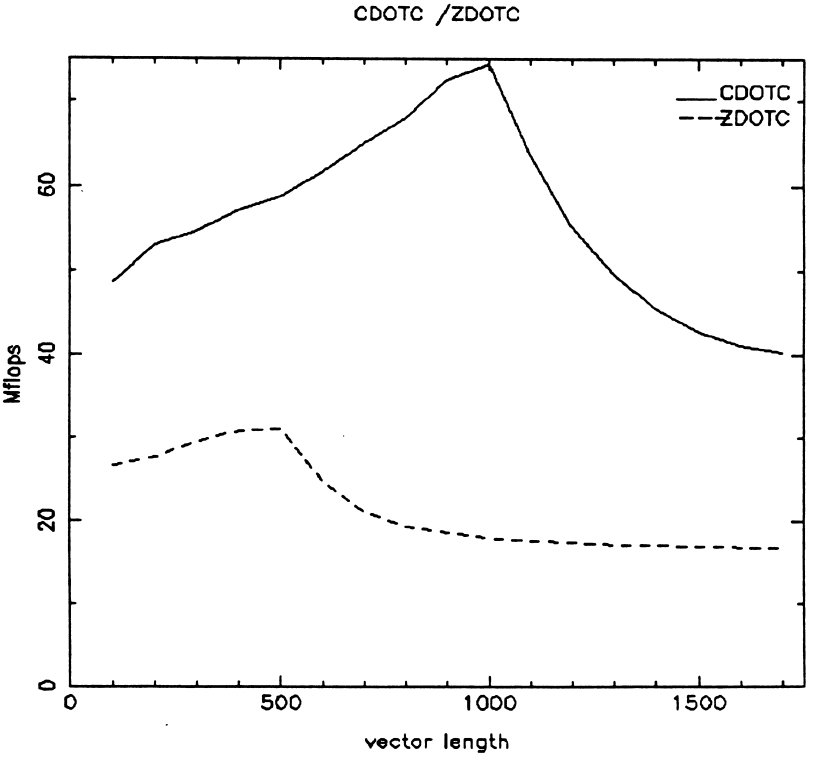
Table D-4. BLAS1 Complex*16 Performance Comparison

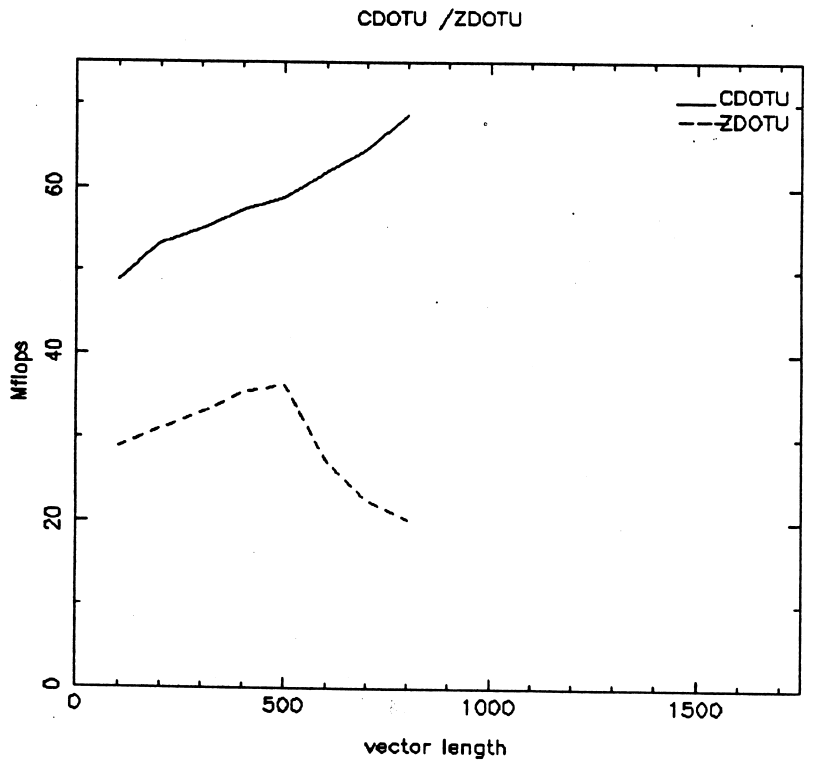
N	DZASUM	ZAXPY	ZSCAL	ZDSCAL	ZDOTC	ZDOTU
100	5.07	20.04	15.92	26.97	26.66	28.93
200	5.20	20.51	16.23	28.13	27.61	31.04
300	5.15	20.62	16.43	28.22	29.48	33.01
400	5.19	20.73	16.51	28.48	30.78	35.43
500	5.20	20.40	16.21	27.65	31.03	36.37
600	4.59	17.13	14.30	21.30	24.75	27.36
700	4.17	15.17	13.00	18.02	21.05	22.55
800	3.95	14.11	12.27	16.19	19.34	20.39
900	3.85	13.62	11.93	15.29	18.55	
1000	3.79	13.27	11.69	14.65	17.92	
1100	3.75	13.05	11.57	14.28	17.61	
1200	3.72	12.90	11.44	14.01	17.37	
1300	3.70	12.78	11.35	13.78	17.15	
1400	3.69	12.71	11.29	13.66	17.06	
1500	3.68	12.65	11.25	13.58	16.97	
1600	3.67	12.61	11.21	13.47	16.88	
1700	3.67	12.58	11.19	13.40	16.86	

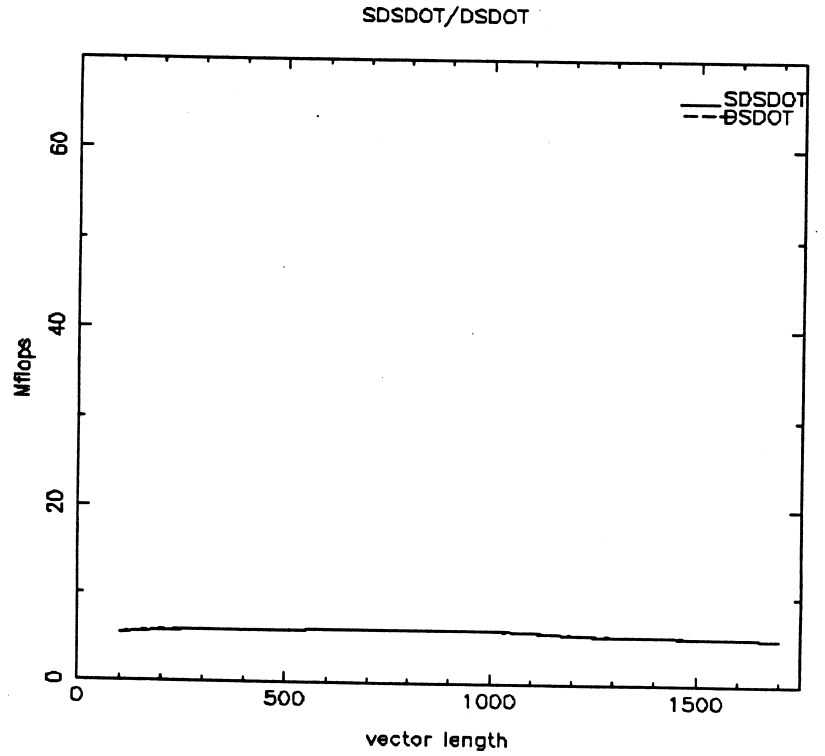

SASUM /DASUM /SCASUM/DZASUM
SASUM
DASUM
SCASUM
DZASUM
Mflops
0
20
40
60
0
500
1000
1500
vector length

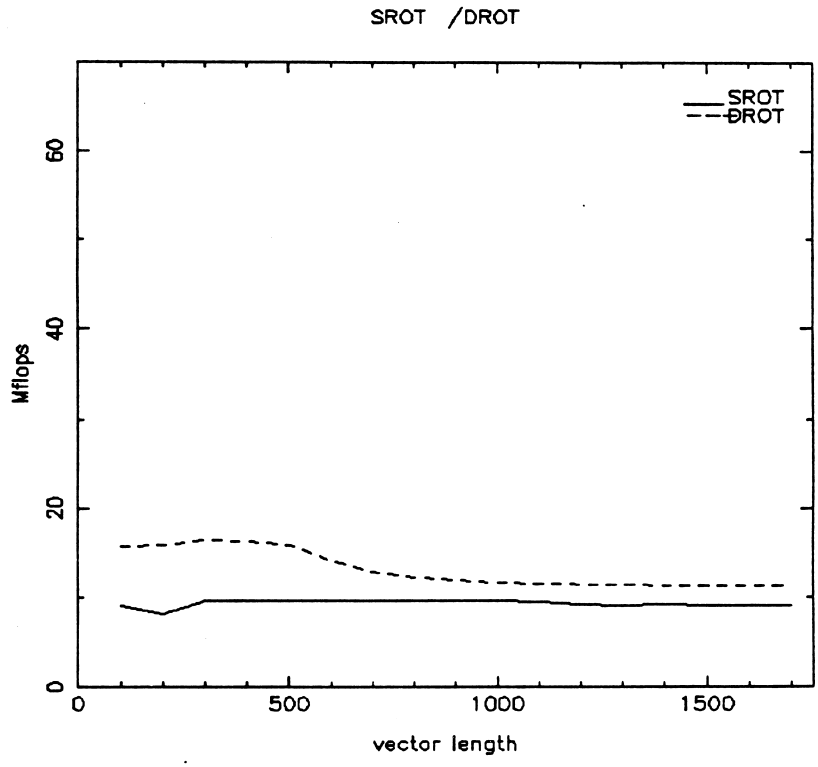

SAXPY /DAXPY /CAXPY /ZAXPY
SAXPY
DAXPY
CAXPY
ZAXPY
Mflops
0
20
40
60
0
500
1000
1500
vector length

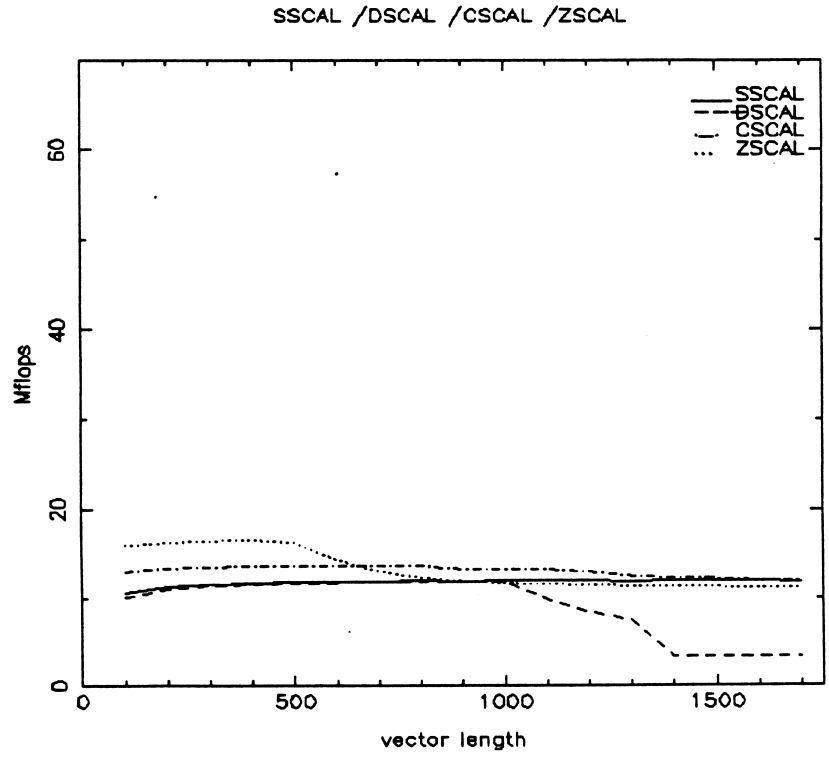

SDOT /DDOT
SDOT
DDOT
Mflops
0
20
40
60
0
500
1000
1500
vector length

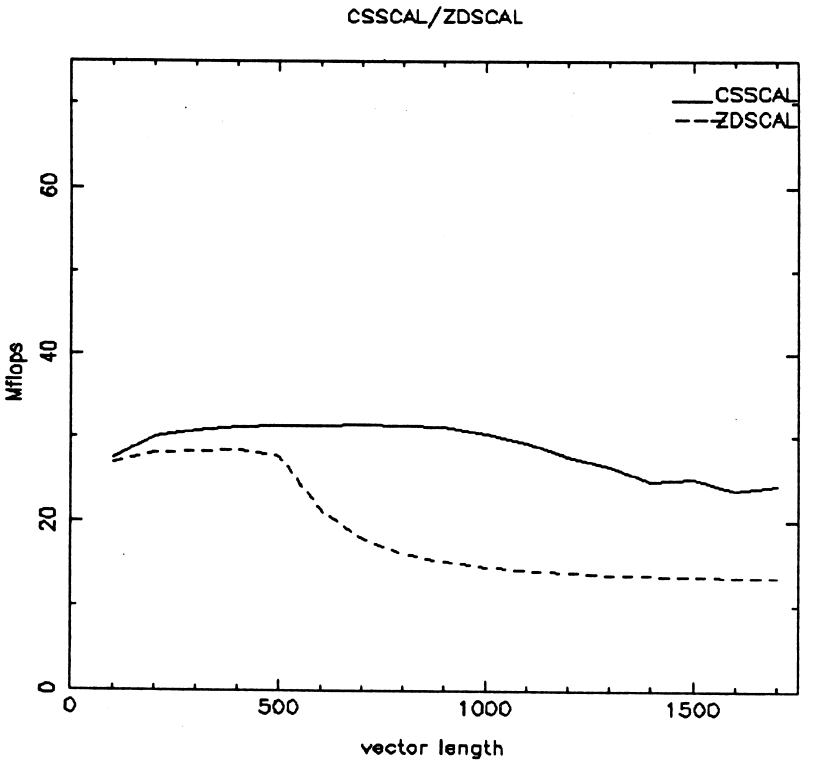

CDOTC /ZDOTC
CDOTC
ZDOTC
Mflops
0
20
40
60
0
500
1000
1500
vector length


CDOTU /ZDOTU
CDOTU
ZDOTU
Mflops
0
20
40
60
0
500
1000
1500
vector length


SDSDOT/DSDOT
SDSDOT
DSDOT
Mflops
0
20
40
60
0
500
1000
1500
vector length


SROT /DROT
SROT
DROT
Mflops
0
20
40
60
0
500
1000
1500
vector length


SSCAL /DSCAL /CSCAL /ZSCAL
SSCAL
DSCAL
CSCAL
ZSCAL
Mflops
0
20
40
60
0
500
1000
1500
vector length



BLAS2 TEST CASES AND PERFORMANCE

The test cases for the BLAS2 routines were adopted from the Public Domain LAPACK BLAS test programs. Each test program is driven by a data file which contains information such as the test ratio threshold value, the values of N, K, stride, ALPHA, BETA, and the routines to be tested.

The test driver was modified so it only tests the unit stride, with ALPHA and BETA equal to non-zero and non-one. The value of N goes up to 512. For each value of N, each routine was called multiple times using different values of K, UPLO, TRANS, and/or DIAG.

Performance was measured by calculating Mflops for each N. The performance for LAPACK's Fortran single and double-precision routines was also measured using the same test drivers.

Table D-5 shows the performance comparison for single-precision routines between KAI and Fortran versions. Table D-6 shows the performance comparison for double-precision routines between KAI and Fortran versions. Table D-7 shows the performance comparison for single-precision complex routines and Table D-8 shows the performance comparison for double-precision complex routines.

Table D-5. BLAS2 Single-Precision Performance Comparison

N	SGEMV		SGBMV		SSYMV		SSBMV		SSPMV	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	5.60	6.72	1.87	1.83	2.73	7.70	2.32	2.26	6.78	7.47
50	11.33	8.95	2.60	2.52	5.89	12.29	3.15	3.02	12.60	13.15
100	23.87	9.62	2.93	2.89	11.03	13.82	3.51	3.38	14.28	14.35
200	36.80	9.92	3.09	3.05	19.07	14.85	3.78	3.57	15.09	15.12
300	44.05	9.82	3.15	3.10	24.60	14.86	3.80	3.59	15.13	15.10
512	48.57	9.37	3.21	3.13	32.88	13.86	3.82	3.53	14.50	14.43

N	STRMV		STBMV		STPMV		STRSV		STBSV	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	2.70	4.38	1.61	1.50	5.02	4.48	1.83	2.82	0.63	0.63
50	6.78	7.39	2.19	2.10	8.03	7.99	4.88	5.66	0.80	0.81
100	13.59	8.37	2.47	2.42	8.83	8.68	9.95	7.04	0.87	0.90
200	23.81	8.94	2.68	2.62	9.19	9.07	18.08	8.13	0.90	0.94
300	30.78	9.07	2.73	2.66	9.28	9.20	24.27	8.50	0.92	0.95
512	40.56	9.04	2.76	2.65	9.22	9.14	33.32	8.67	0.93	0.95

N	STPSV		SGER		SSYR		SSPR		SSYR2		SSPR2	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	2.83	2.83	5.63	6.58	3.15	4.78	4.93	4.97	7.50	8.09	8.05	8.13
50	5.65	5.85	9.95	8.11	6.67	7.17	7.95	7.60	12.59	12.65	13.31	13.57
100	7.20	7.23	12.41	8.15	9.02	7.69	8.19	7.87	13.99	13.89	14.52	14.49
200	8.21	8.23	13.74	8.21	10.84	8.01	8.40	8.19	14.59	14.77	14.91	14.98
300	8.58	8.58	14.31	8.07	11.68	8.10	8.45	8.25	14.53	14.67	14.86	14.89
512	8.77	8.76	14.91	7.99	13.95	7.97	8.33	8.15	13.58	13.71	14.22	14.11

Table D-6. BLAS2 Double-Precision Performance Comparison

N	DGEMV		DGBMV		DSYMV		DSBMV		DSPMV	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	6.73	5.91	1.90	1.77	3.42	6.92	2.26	2.40	6.89	7.30
50	15.90	7.00	2.63	2.49	7.80	9.75	2.86	3.27	10.57	10.47
100	21.98	7.57	2.95	2.88	12.46	10.81	3.59	3.60	11.08	10.97
200	25.15	7.45	3.04	2.93	16.72	10.56	3.79	3.72	10.73	10.79
300	26.32	7.16	3.11	2.97	18.69	10.18	3.81	3.69	10.34	10.19
512	26.97	5.05	3.17	3.01	21.93	6.15	3.78	3.55	9.49	9.26

N	DTRMV		DTBMV		DTPMV		DTRSV		DTBSV	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	2.39	4.43	1.59	1.51	4.76	4.67	2.01	2.61	0.56	0.59
50	5.35	6.61	2.21	2.19	7.08	7.11	5.37	4.54	0.71	0.76
100	9.12	7.32	2.59	2.52	7.51	7.40	9.80	5.77	0.77	0.82
200	13.75	7.53	2.77	2.58	7.82	7.68	14.86	6.60	0.79	0.84
300	16.69	7.51	2.84	2.63	7.69	7.63	17.85	6.81	0.81	0.85
512	20.09	6.94	2.84	2.63	7.38	7.30	21.10	6.48	0.81	0.85

N	DTPSV		DGER		DSYR		DSPR		DSYR2		DSPR2	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	2.64	2.67	5.66	6.50	3.61	4.85	4.69	5.17	7.99	7.87	7.88	8.30
50	4.83	4.86	8.63	6.86	6.24	6.12	6.82	6.86	10.17	10.19	11.24	11.03
100	6.02	6.01	10.00	6.59	8.58	6.64	6.89	6.84	10.75	10.05	11.07	10.95
200	6.69	6.70	10.85	6.37	9.82	6.61	6.92	6.89	10.20	10.11	10.41	10.39
300	6.84	6.89	11.16	5.86	10.52	6.54	6.66	6.62	9.75	9.71	9.82	9.83
512	6.80	6.77	11.31	6.41	11.05	6.17	6.25	6.21	6.74	6.79	8.97	8.96

Table D-7. BLAS2 Single-Precision Performance Comparison For Complex Routines:

N	CGEMV	CGBMV	CHEMV	CHBMV	CHPMV	CTRMV	CTBMV	CTPMV
20	11.53	5.85	12.54	5.99	12.59	4.46	4.35	10.70
50	29.05	7.40	15.41	6.91	15.87	14.26	5.90	12.83
100	42.09	7.99	16.26	8.21	16.36	28.15	6.55	13.20
200	52.71	8.13	16.69	8.38	16.71	42.31	6.84	13.42
300	57.23	8.25	16.58	8.37	16.65	49.32	6.85	13.42
512	59.60	8.38	13.96	8.24	15.91	56.06	6.84	13.20

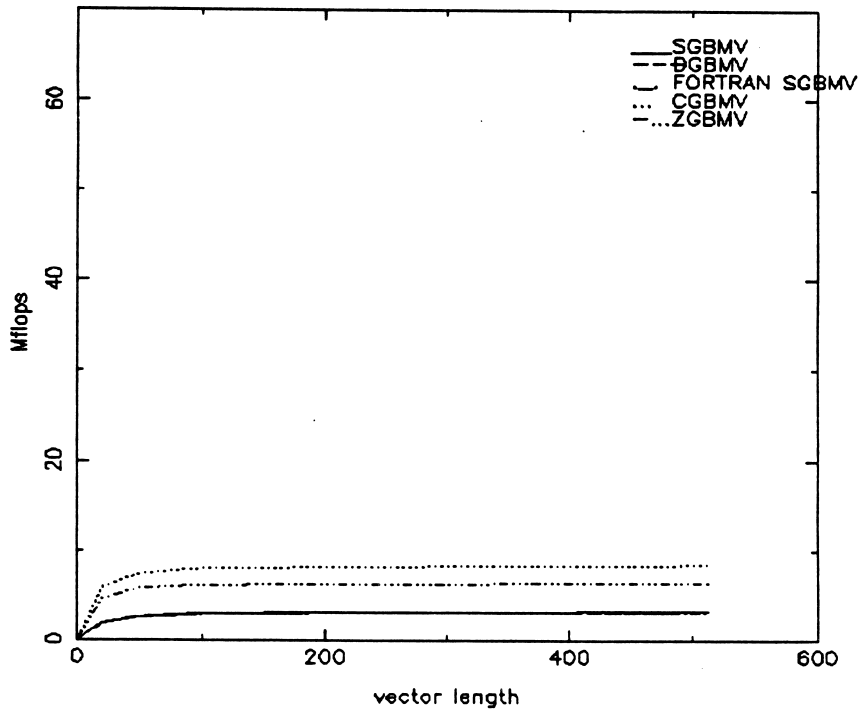
N	CTRSV	CTBSV	CTPSV	CGERC	CGERU	CHER	CHPR	CHER2	CHPR2
20	2.93	1.72	5.86	19.92	15.01	11.23	11.71	13.18	13.28
50	9.15	2.02	9.35	28.11	26.28	15.56	15.41	15.48	15.94
100	17.95	2.17	11.08	29.80	29.84	17.39	17.64	16.99	17.21
200	29.42	2.23	12.16	30.42	30.34	18.02	18.03	17.34	17.44
300	36.91	2.23	12.51	30.77	30.84	17.94	18.03	17.20	17.25
512	45.89	2.23	12.65	30.94	30.98	11.52	17.57	14.41	16.35

Table D-8. BLAS2 Double-Precision Performance Comparison For Complex Routines

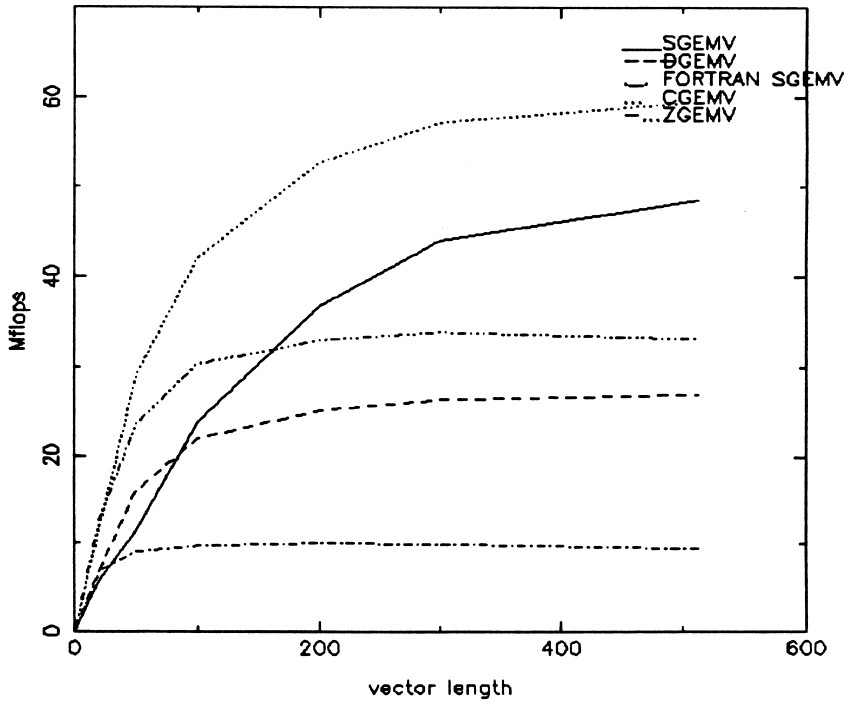
N	ZGEMV	ZGBMV	ZHEMV	ZHBMV	ZHPMV	ZTRMV	ZTBMV	ZTPMV
20	12.52	4.55	9.14	5.16	9.29	4.06	3.82	7.89
50	23.60	5.79	10.33	5.91	10.05	14.37	5.04	8.55
100	30.35	6.14	10.36	6.09	10.31	23.26	5.57	8.65
200	32.97	6.24	10.18	6.30	10.12	29.53	5.67	8.55
300	33.86	6.30	9.90	6.32	9.77	31.75	5.62	8.34
512	33.16	6.37	8.28	6.32	9.39	32.56	5.69	7.98

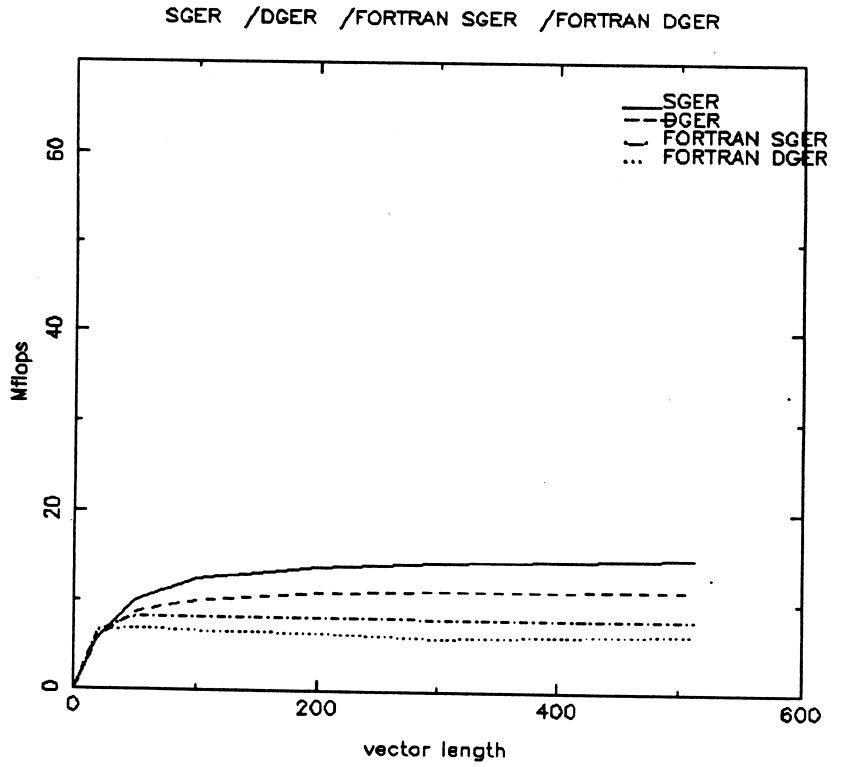
N	ZTRSV	ZTBSV	ZTPSV	ZGERC	ZGERU	ZHER	ZHPR	ZHER2	ZHPR2
20	3.64	1.46	4.36	14.28	14.23	7.67	8.18	9.28	9.94
50	9.35	1.74	6.26	18.28	17.32	9.24	8.95	10.44	10.47
100	16.94	1.81	7.24	19.34	19.66	9.48	9.64	10.53	10.67
200	23.42	1.84	7.68	19.83	19.83	9.30	9.34	10.33	10.33
300	26.61	1.85	7.68	19.83	19.85	9.03	8.99	10.15	10.05
512	28.60	1.86	7.52	19.94	19.97	6.79	8.47	8.23	9.69

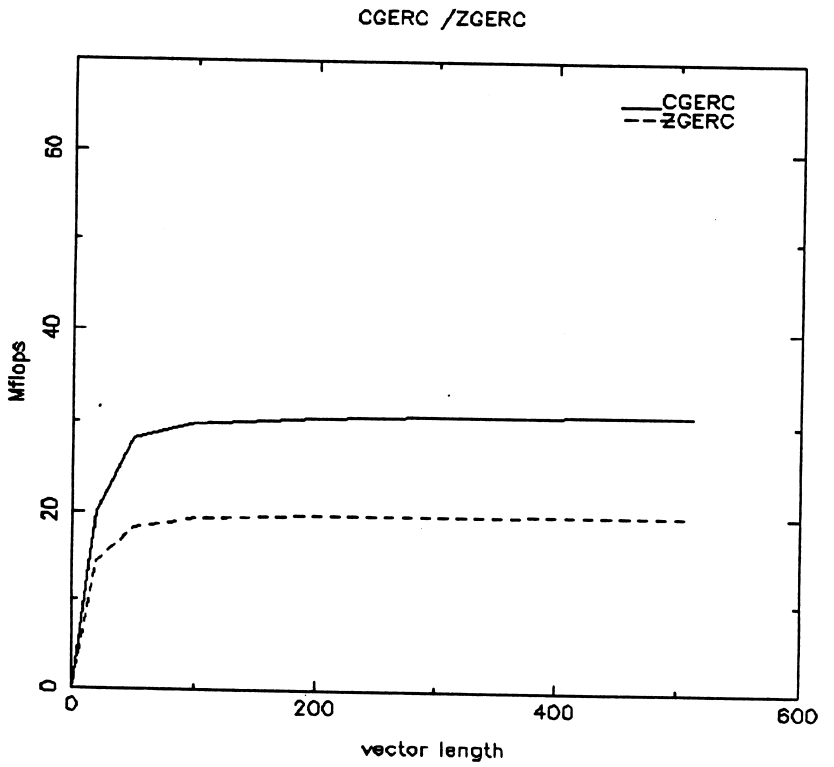
SGBMV /DGBMV /CGBMV /ZGBMV /FORTRAN SGBMV



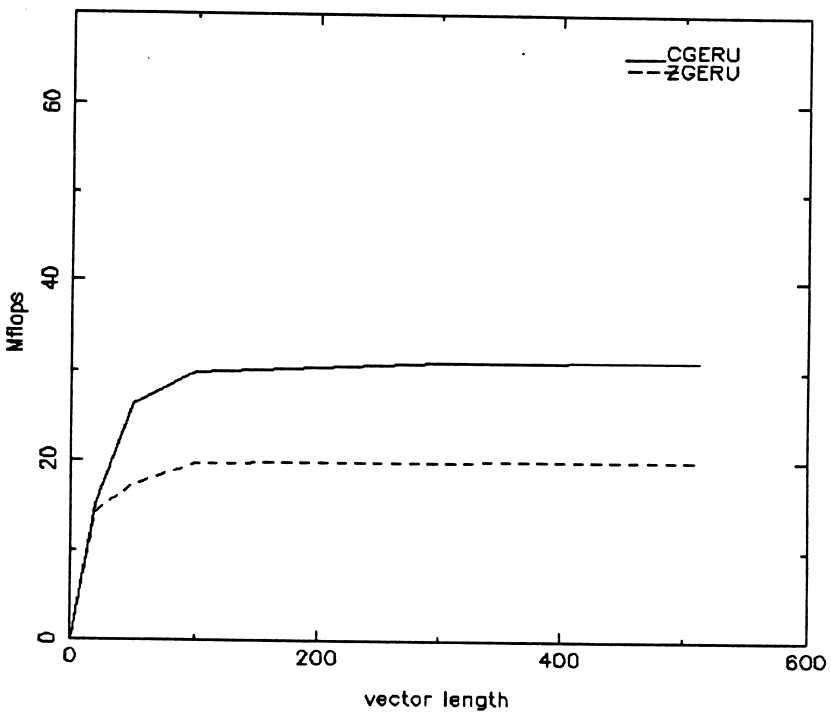
SGEMV /DGEMV /CGEMV /ZGEMV /FORTRAN SGEMV



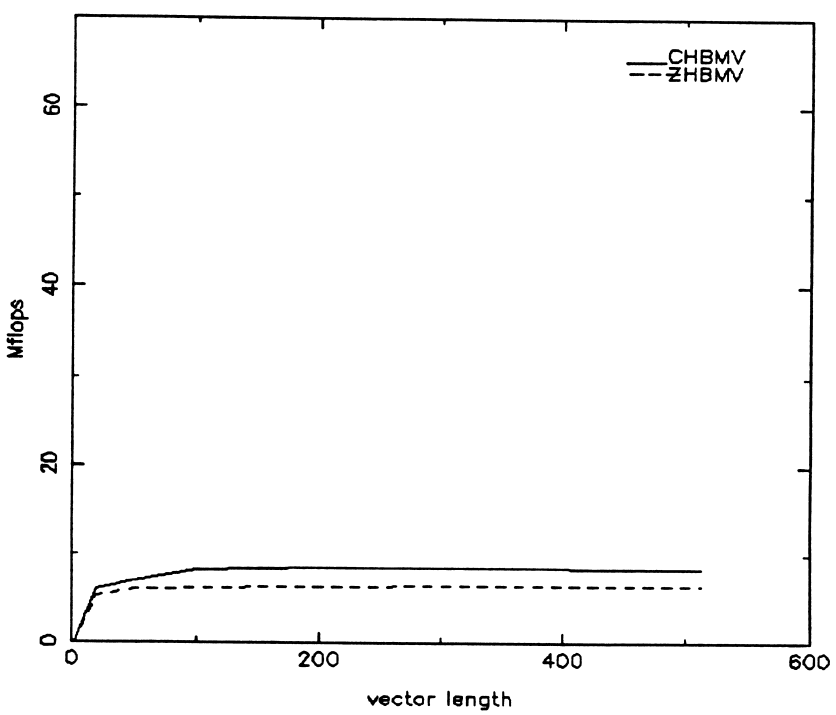

SGER /DGER /FORTRAN SGER /FORTRAN DGER
SGER
DGER
FORTRAN SGER
FORTRAN DGER
Mflops
0
20
40
60
0
200
400
600
vector length

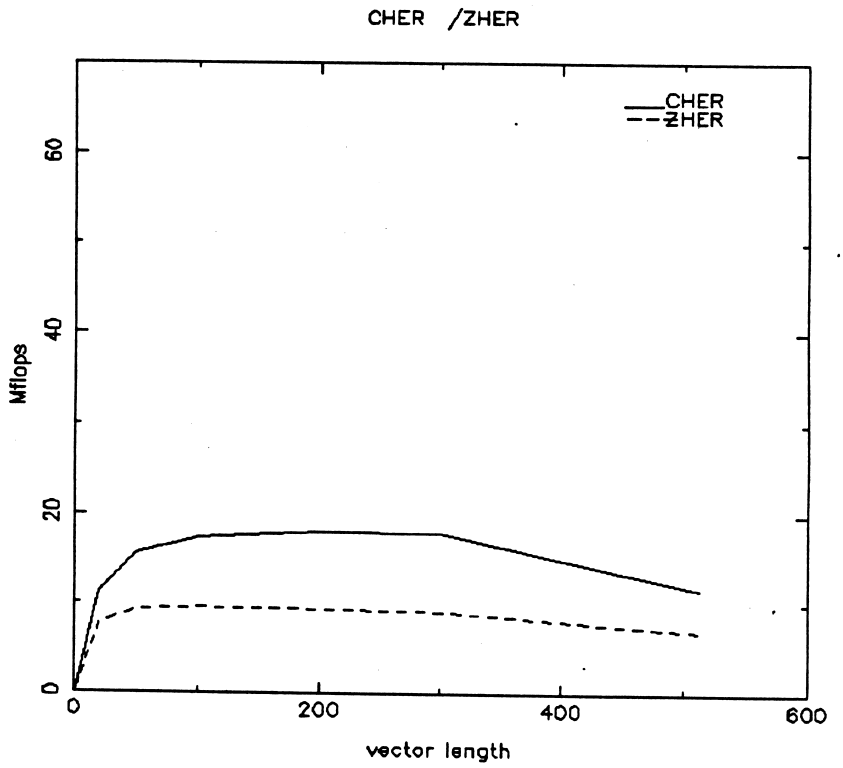

CGERC /ZGERC
CGERC
ZGERC
Mflops
0
20
40
60
0
200
400
600
vector length

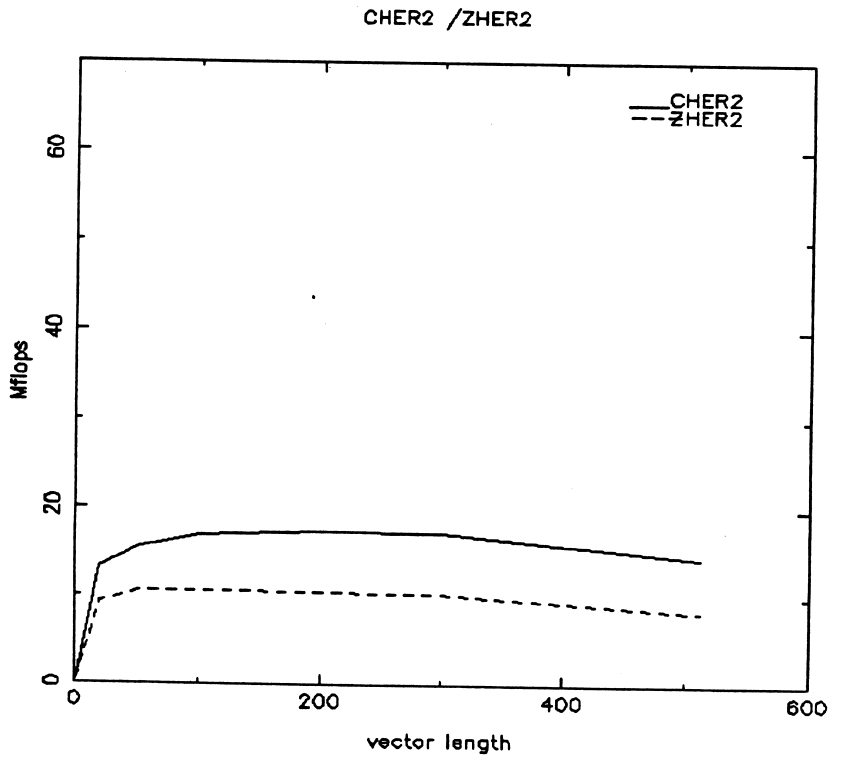
CGERU /ZGERU



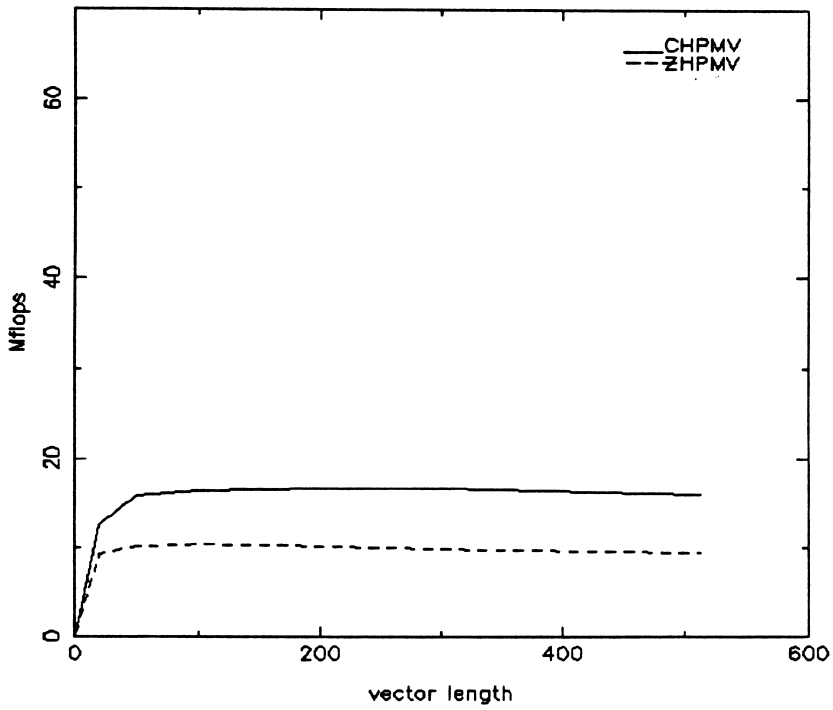
CHBMV /ZHBMV



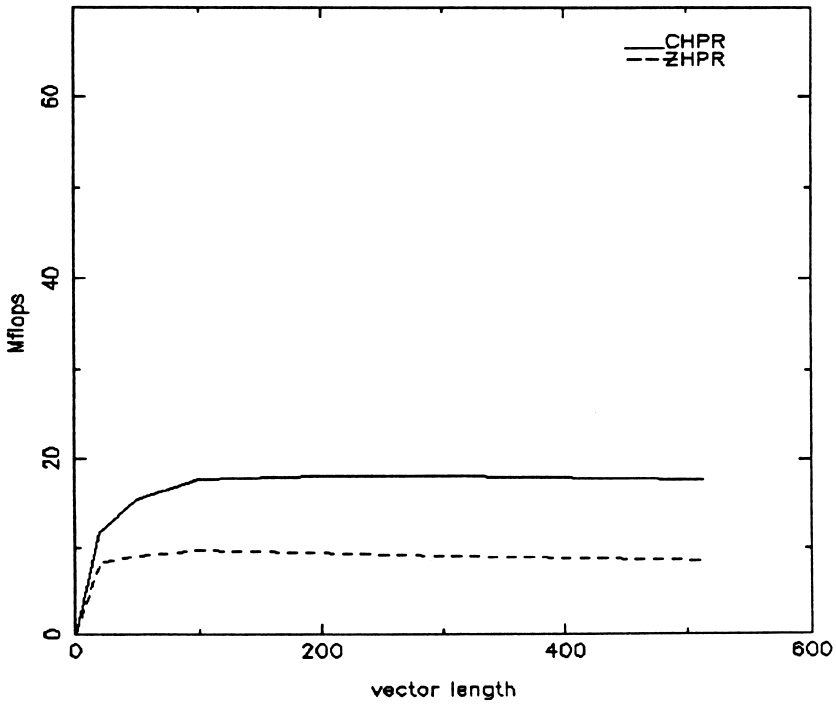

CHER /ZHER
CHER
ZHER
Mflops
0
20
40
60
0
200
400
600
vector length

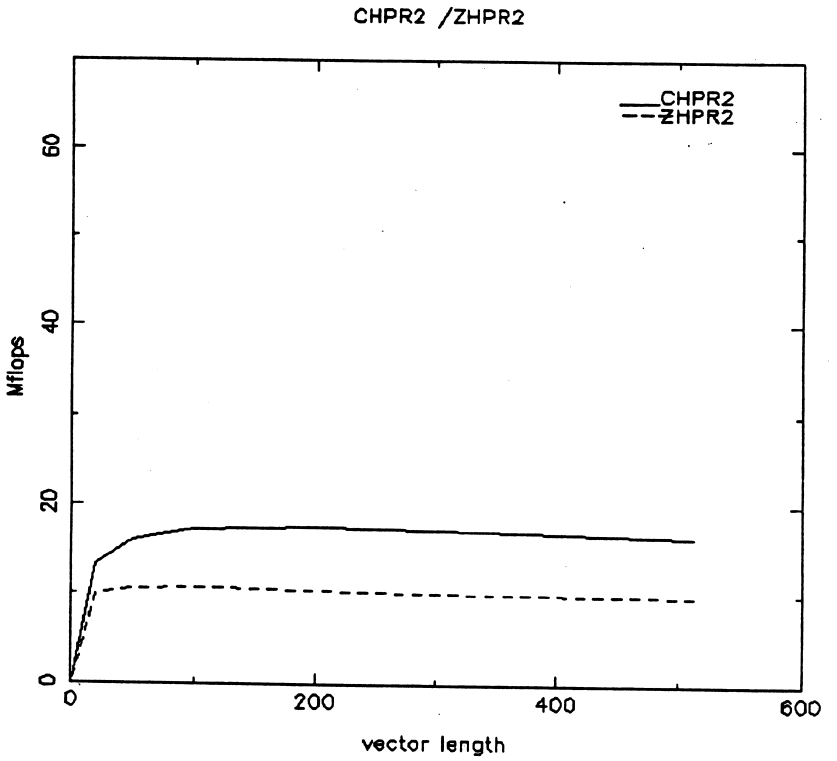

CHER2 /ZHER2
CHER2
ZHER2
Mflops
0
20
40
60
0
200
400
600
vector length

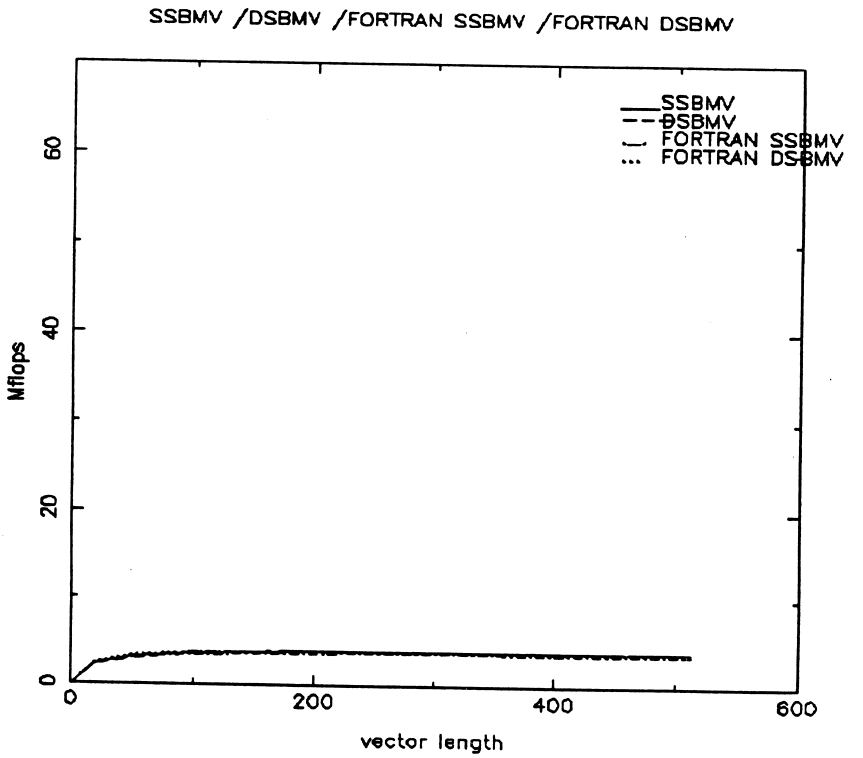
CHPMV /ZHPMV

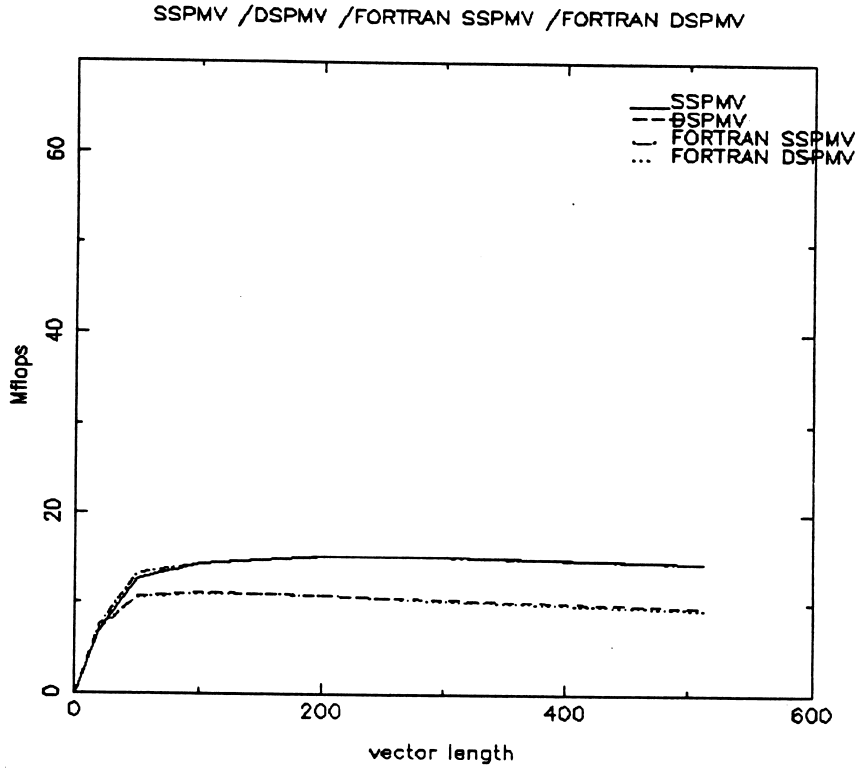


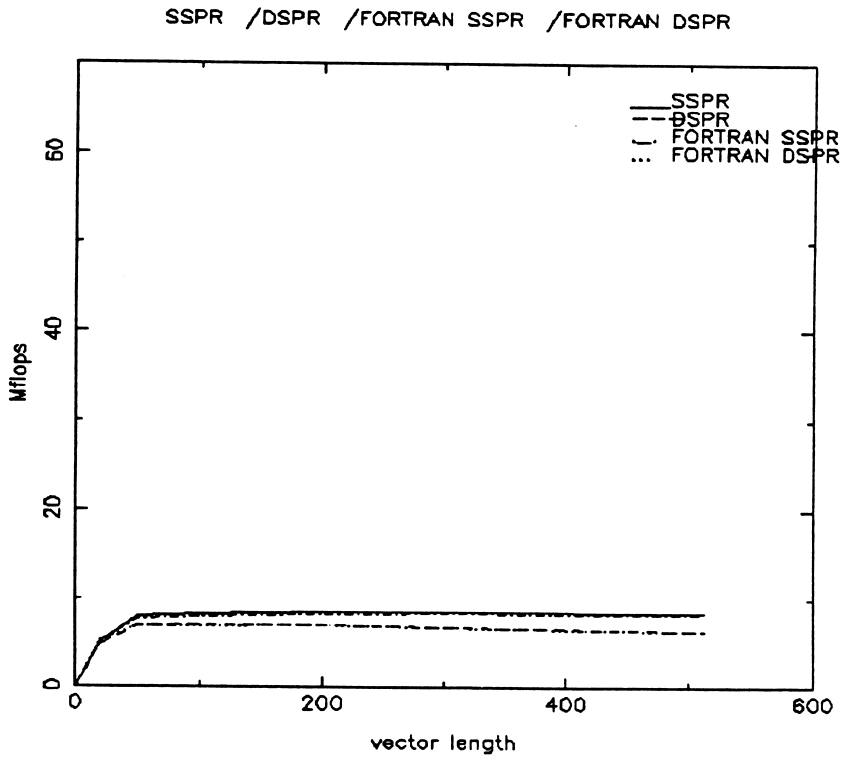
CHPR /ZHPR



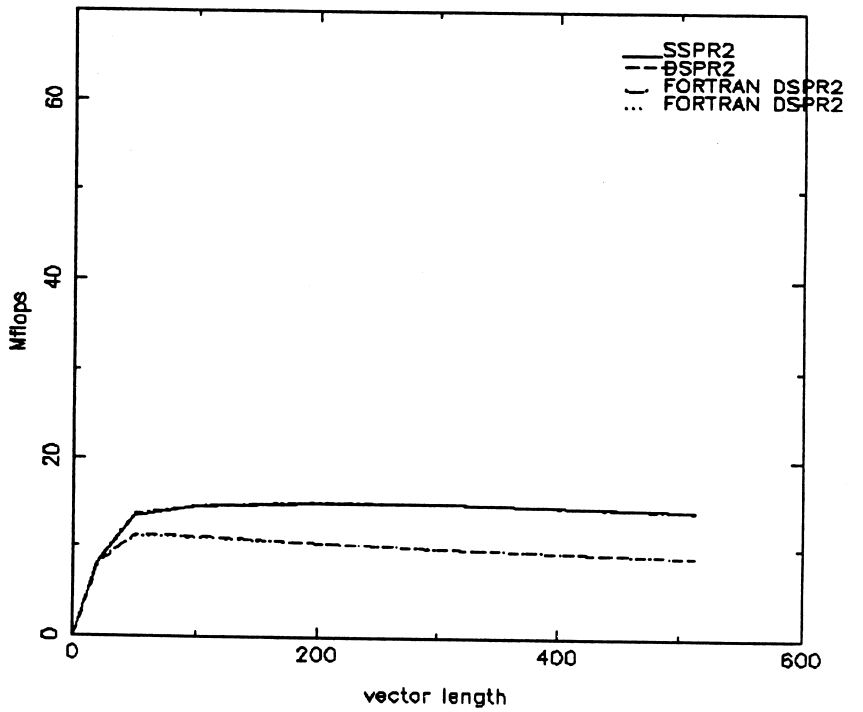

CHPR2 /ZHPR2
CHPR2
ZHPR2
Mflops
0
20
40
60
0
200
400
600
vector length


SSBMV /DSBMV /FORTRAN SSBMV /FORTRAN DSBMV
SSBMV
DSBMV
FORTRAN SSBMV
FORTRAN DSBMV
Mflops
0
20
40
60
0
200
400
600
vector length

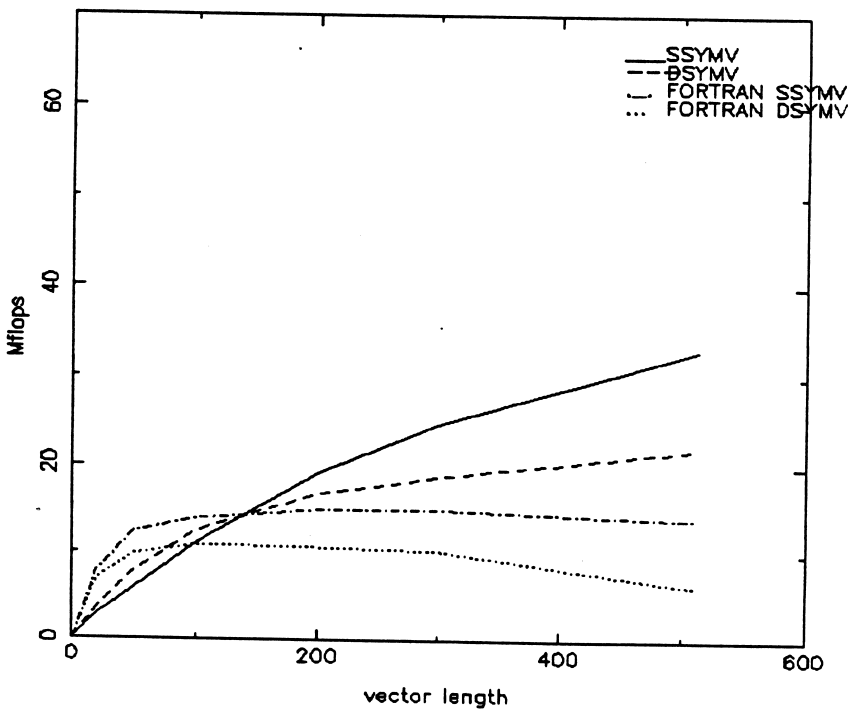

SSPMV /DSPMV /FORTRAN SSPMV /FORTRAN DSPMV
SSPMV
DSPMV
FORTRAN SSPMV
FORTRAN DSPMV
Mflops
0
20
40
60
0
200
400
600
vector length


SSPR /DSPR /FORTRAN SSPR /FORTRAN DSPR
SSPR
DSPR
FORTRAN SSPR
FORTRAN DSPR
Mflops
0
20
40
60
0
200
400
600
vector length

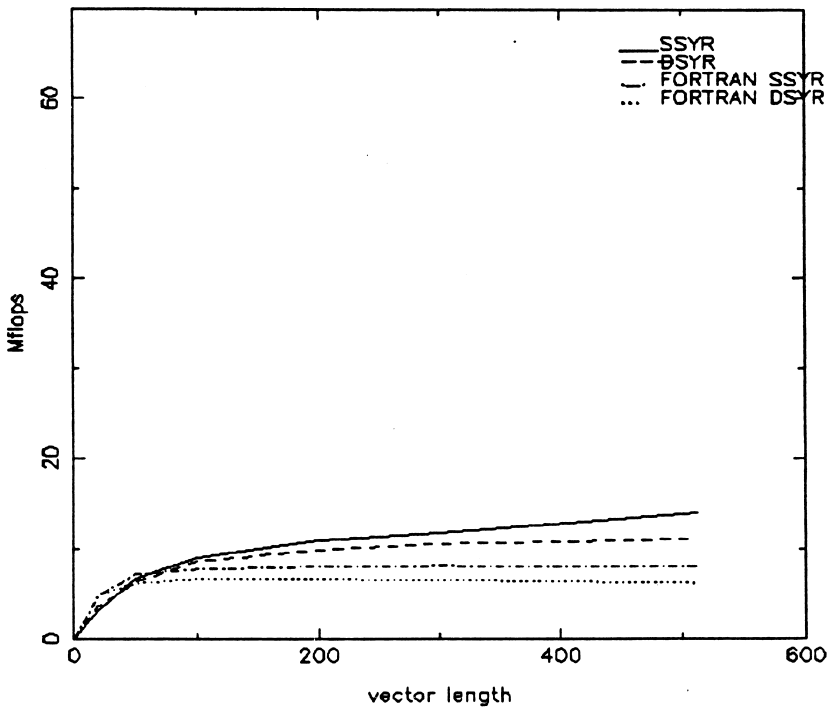
SSPR2 /DSPR2 /FORTRAN DSPR2 /FORTRAN DSPR2



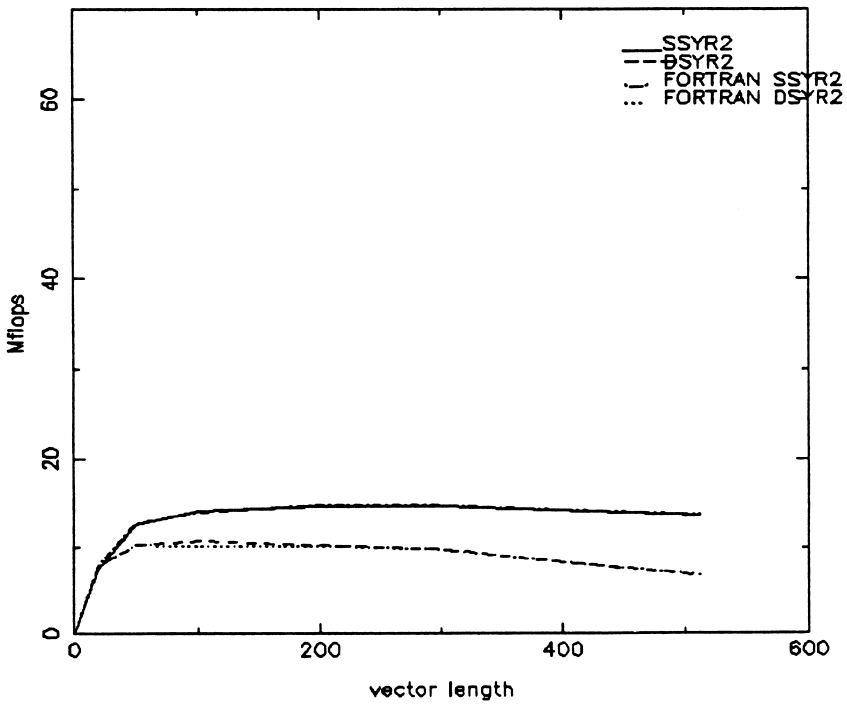
SSYMV /DSYMV /FORTRAN SSYMV /FORTRAN DSYMV

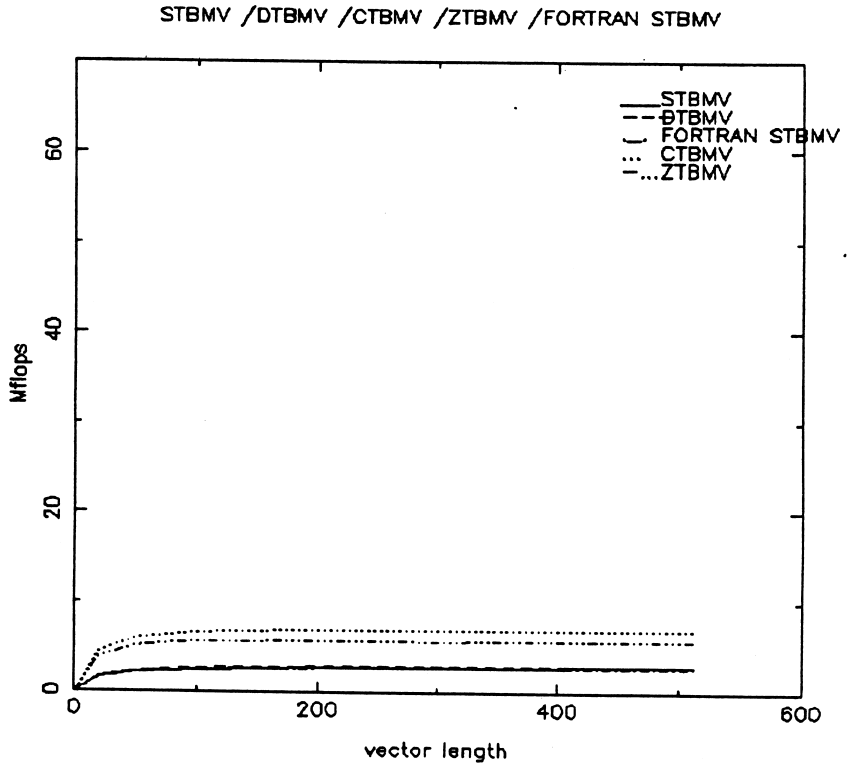


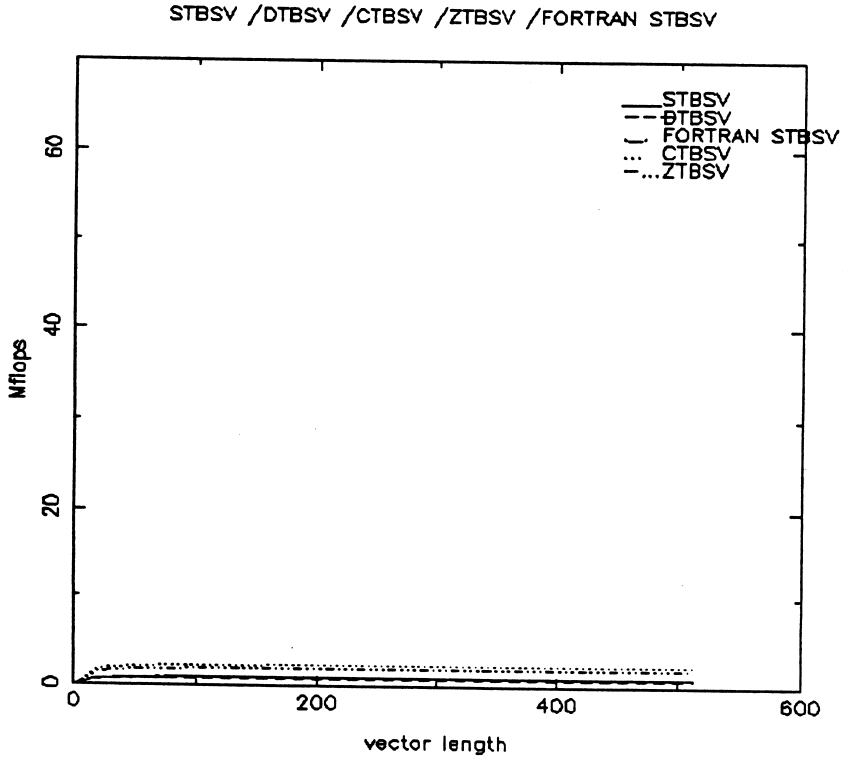
SSYR /DSYR /FORTRAN SSYR /FORTRAN DSYR

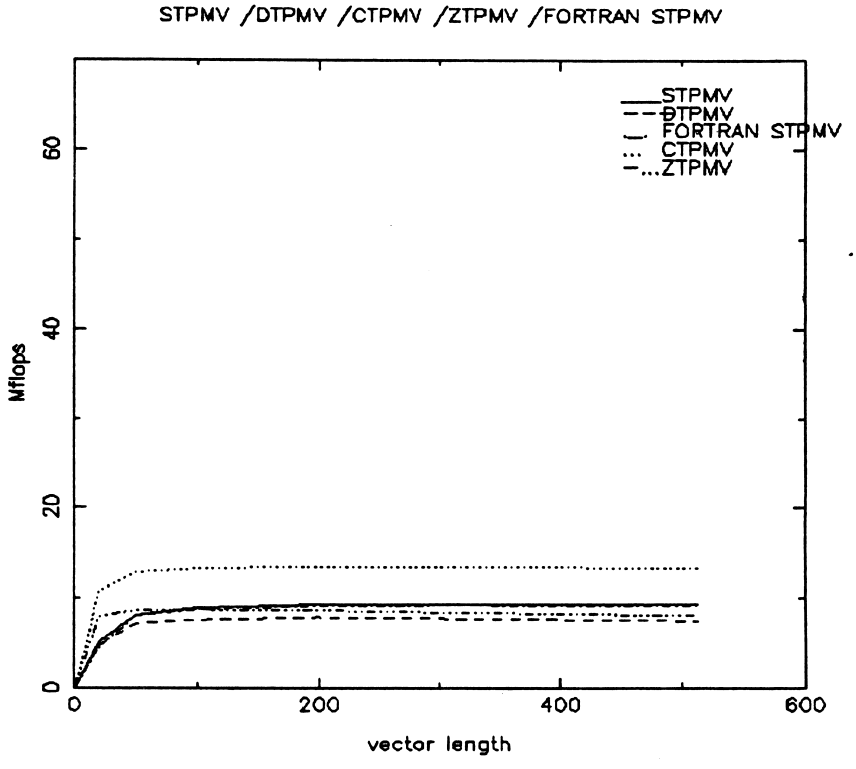


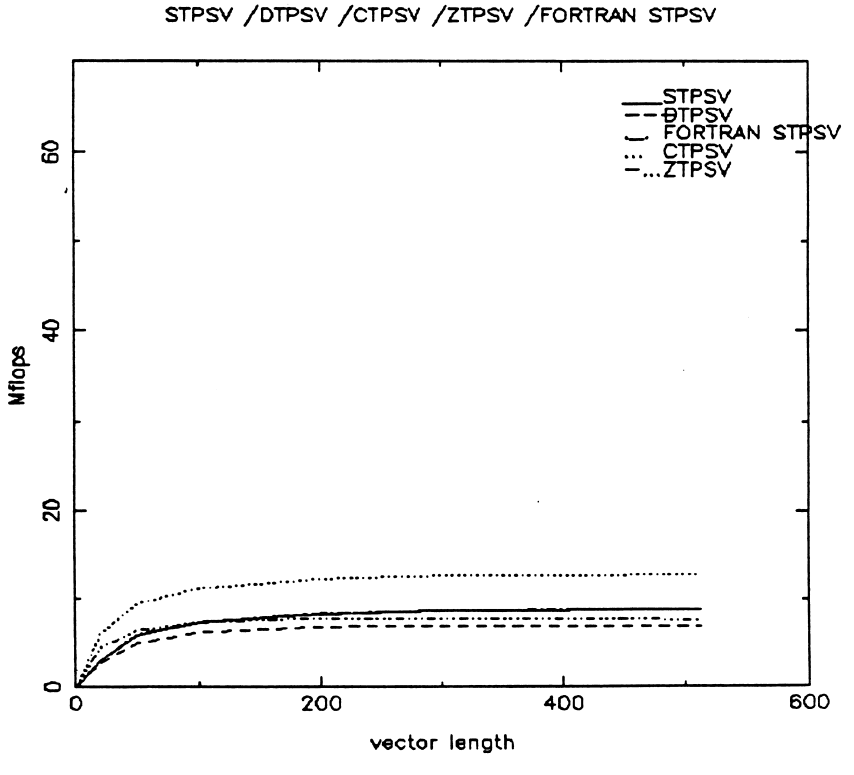
SSYR2 /DSYR2 /FORTRAN SSYR2 /FORTRAN DSYR2

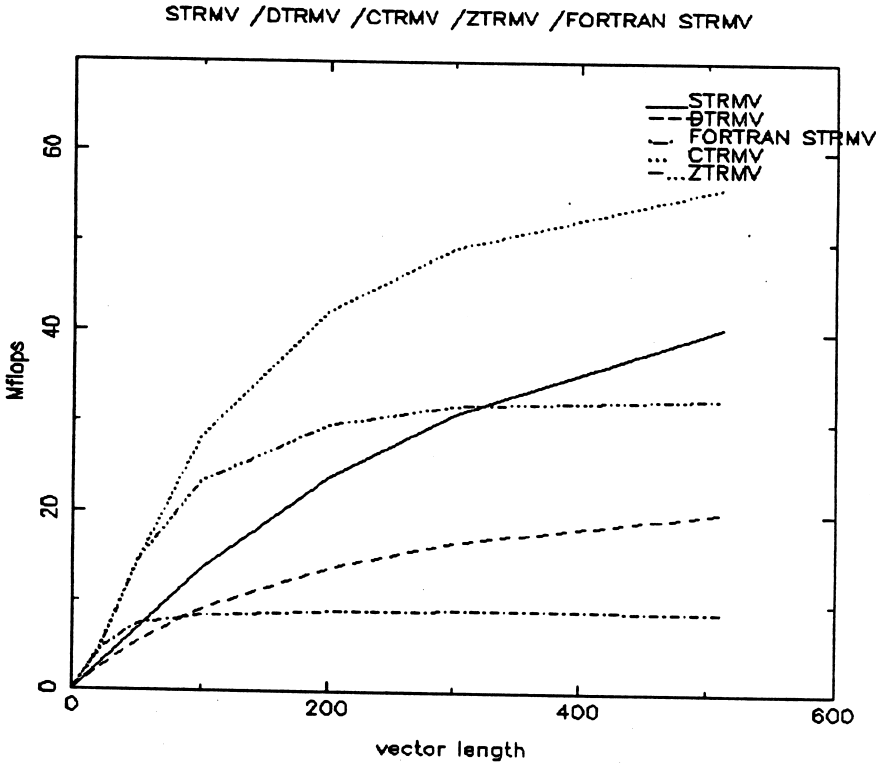


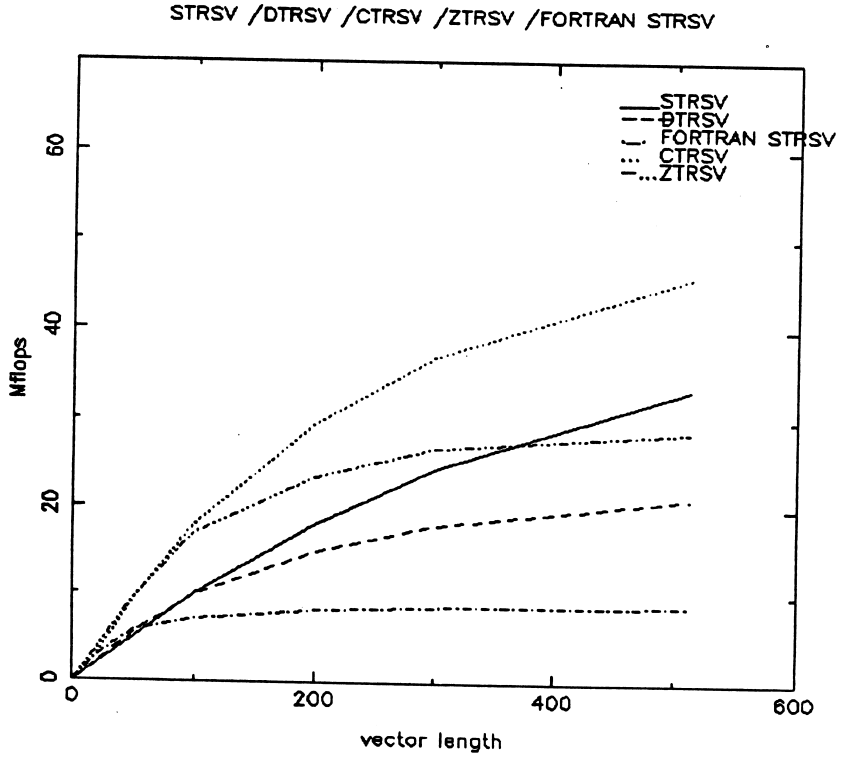

STBMV /DTBMV /CTBMV /ZTBMV /FORTRAN STBMV
STBMV
DTBMV
FORTRAN STBMV
CTBMV
ZTBMV
Mflops
0
20
40
60
0
200
400
600
vector length


STBSV /DTBSV /CTBSV /ZTBSV /FORTRAN STBSV
STBSV
DTBSV
FORTRAN STBSV
CTBSV
ZTBSV
Mflops
0
20
40
60
0
200
400
600
vector length


STPMV /DTPMV /CTPMV /ZTPMV /FORTRAN STPMV
STPMV
DTPMV
FORTRAN STPMV
CTPMV
ZTPMV
Mflops
0
20
40
60
0
200
400
600
vector length


STPSV /DTPSV /CTPSV /ZTPSV /FORTRAN STPSV
STPSV
DTPSV
FORTRAN STPSV
CTPSV
ZTPSV
Mflops
0
20
40
60
0
200
400
600
vector length


STRMV /DTRMV /CTRMV /ZTRMV /FORTRAN STRMV
STRMV
DTRMV
FORTRAN STRMV
CTRMV
ZTRMV
Mflops
0
20
40
60
0
200
400
600
vector length


STRSV /DTRSV /CTRSV /ZTRSV /FORTRAN STRSV
STRSV
DTRSV
FORTRAN STRSV
CTRSV
ZTRSV
Mflops
0
20
40
60
0
200
400
600
vector length

BLAS3 TEST CASES AND PERFORMANCE

The test cases for BLAS3 routines were adopted from the Public Domain LAPACK BLAS test programs. Each test program is driven by a data file which contains information such as the threshold value test ratio, the values of N, ALPHA, BETA, and the routines to be tested.

The test driver was modified so that ALPHA and BETA are equal to non-zero and non-one. The value of N goes up to 512. M, N, and/or K are the same value. For each value of N, each routine was called multiple times using different values of UPLO, SIDE, TRANS, and/or DIAG.

Performance was measured by calculating Mflops for each N. The performance for LAPACK's Fortran single- and double-precision routines was measured using the same test drivers.

Table D-9 shows the performance comparison for single-precision routines between KAI and Fortran versions. Table D-10 shows the performance comparison for double-precision routines between KAI and Fortran versions. Table D-11 shows the performance comparison for single-precision complex routines and Table D-12 shows the performance comparison for double-precision complex routines.

Table D-9. BLAS3 Performance Comparison For Single-Precision Routines

N	SGEMM		SSYMM		STRMM		STRSM		SSYRK		SSYR2K	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	5.65	8.69	3.08	8.59	3.18	5.93	2.63	4.29	5.06	7.94	4.74	11.52
50	15.05	9.54	10.41	10.56	10.40	7.08	9.59	5.69	11.52	10.06	11.83	12.69
100	26.03	9.33	18.02	10.53	18.90	6.81	17.74	5.83	19.08	9.62	19.94	12.87
200	39.55	8.82	27.53	10.95	28.70	6.93	27.58	6.09	26.64	9.84	27.48	13.16
300	40.69	6.24	32.73	11.01	33.71	6.92	32.95	6.15	29.99	9.89	30.39	12.93
512	36.76		37.50		39.39		38.78		30.54		30.83	

Table D-10. BLAS3 Performance Comparison For Double-Precision Routines

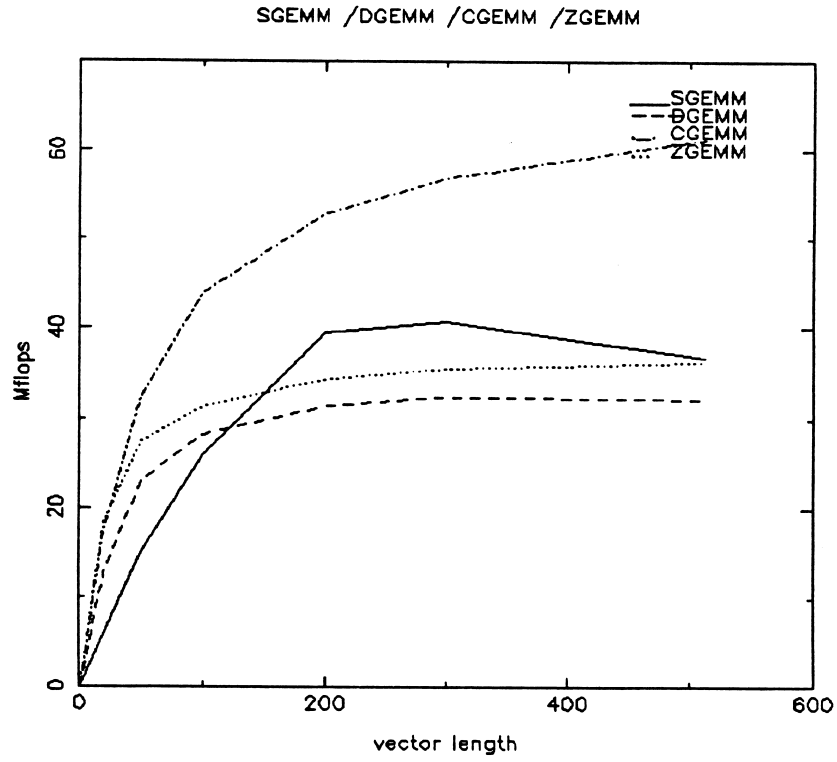
N	DGEMM		DSYMM		DTRMM		DTRSM		DSYRK		DSYR2K	
	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR	KAI	FOR
20	12.74	7.66	5.71	7.81	4.26	5.55	3.61	3.77	8.58	7.25	8.37	10.53
50	23.02	6.86	13.46	8.00	14.86	5.63	13.07	4.39	15.95	7.27	15.72	9.18
100	28.19	7.08	20.23	8.34	22.36	5.54	20.57	4.58	23.07	7.20	23.08	9.44
200	31.37	5.51	25.19	8.40	27.06	5.54	25.54	4.72	26.79	7.23	26.75	9.11
300	32.46	3.90	27.36	8.27	28.64	5.34	27.30	4.62	27.49	7.13	27.37	8.52
512	32.15		28.49		30.00		29.06		28.16		28.00	

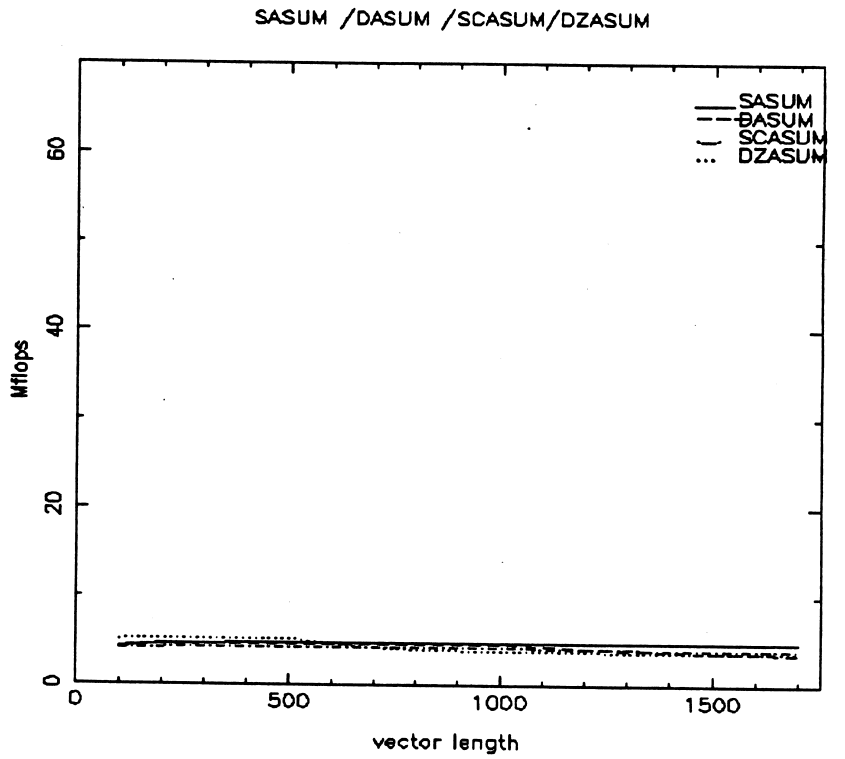
Table D-11. BLAS3 Performance Comparison For Single-Precision Complex Routines

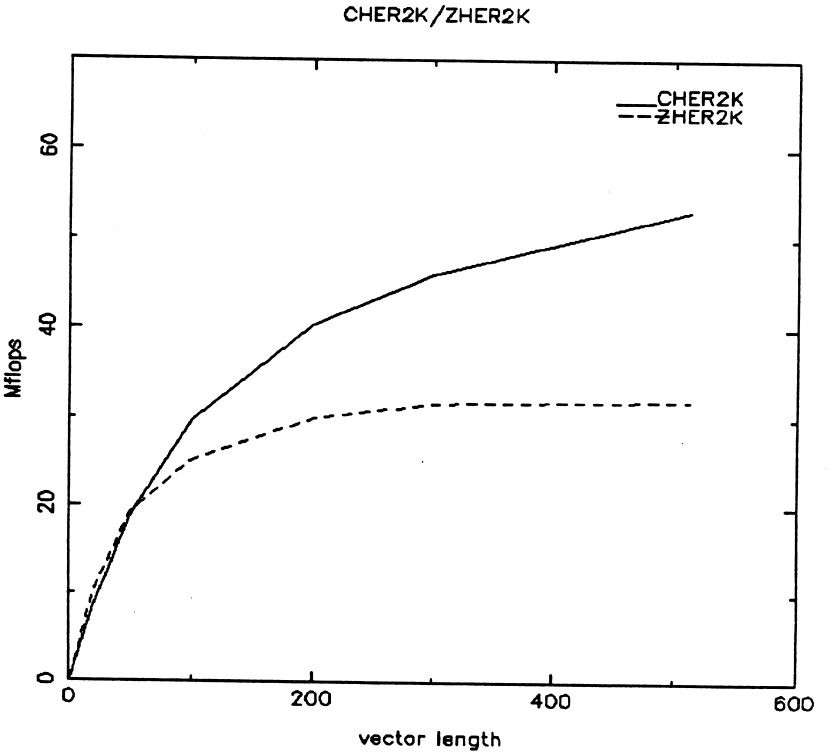
N	CGEMM	CHEMM	CSYMM	CTRMM	CTRSM	CHERK	CSYRK	CHER2K	CSYR2K
20	17.63	7.28	7.26	8.03	6.04	9.18	8.28	8.48	8.13
50	32.32	23.57	23.41	22.80	17.84	20.10	18.47	18.81	18.45
100	43.89	35.46	35.21	34.95	29.00	31.42	29.42	29.68	29.34
200	52.83	44.77	44.57	45.07	40.09	42.61	40.73	40.42	40.06
300	56.88	49.28	49.09	49.51	45.53	48.08	46.40	46.03	45.64
512	61.31	53.47	53.38	54.84	51.95	54.57	53.64	53.15	52.82

Table D-12. BLAS3 Performance Comparison For Double-Precision Complex Routines

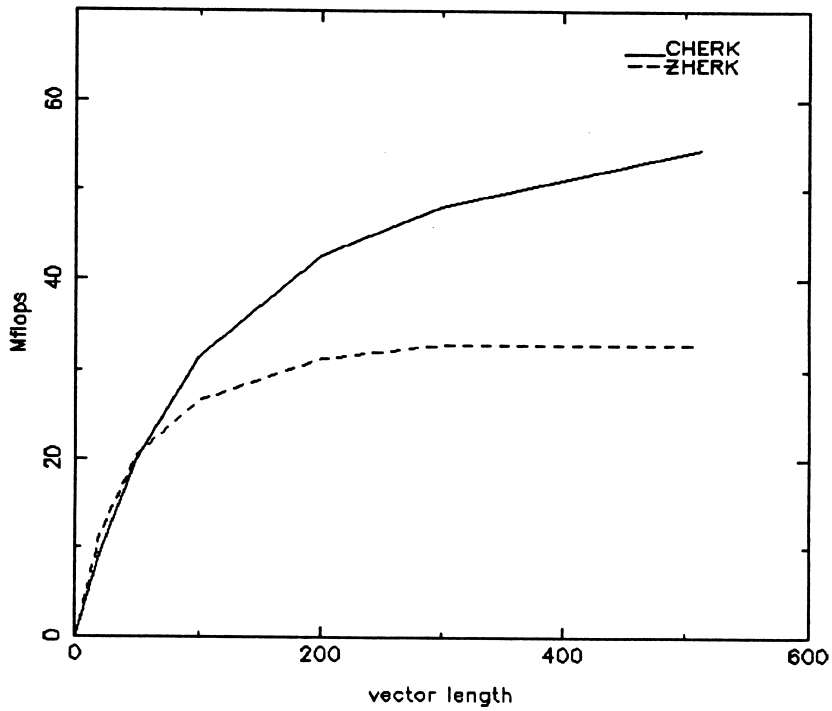
N	ZGEMM	ZHEMM	ZSYMM	ZTRMM	ZTRSM	ZHERK	ZSYRK	ZHER2K	ZSYR2K
20	18.45	9.92	9.79	7.38	6.12	11.37	10.22	10.37	10.29
50	27.38	19.74	19.57	19.31	15.50	20.47	19.22	19.22	19.24
100	31.32	25.15	24.95	26.62	22.82	26.54	25.48	25.15	25.36
200	34.38	28.76	28.64	31.01	28.23	31.27	30.33	29.93	30.31
300	35.54	30.16	30.02	32.66	30.52	32.80	32.01	31.61	32.01
512	36.25	31.20	31.08	33.38	31.81	32.76	32.36	32.01	32.13



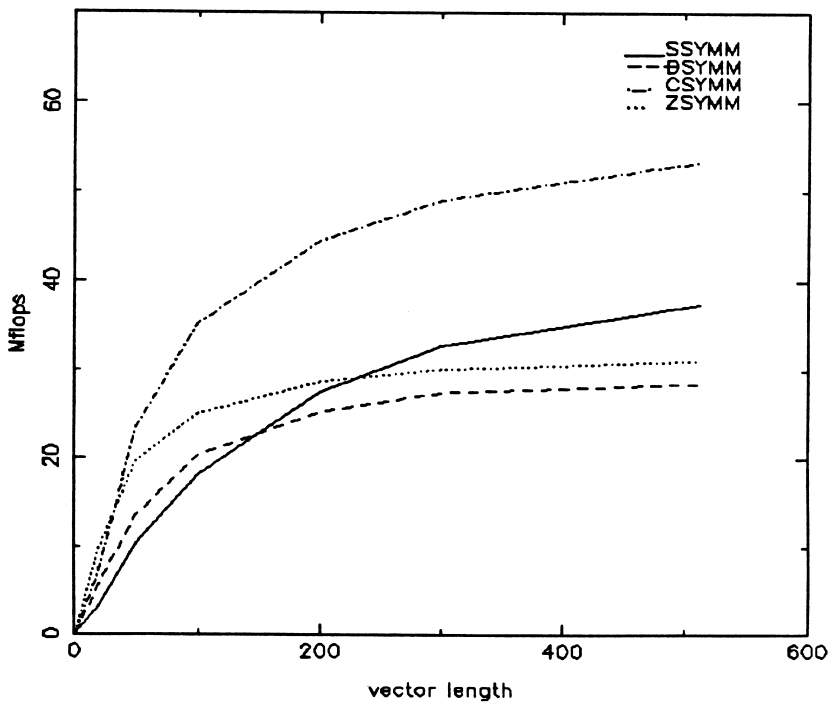

SASUM /DASUM /SCASUM/DZASUM
SASUM
DASUM
SCASUM
DZASUM
Mflops
0
20
40
60
0
500
1000
1500
vector length

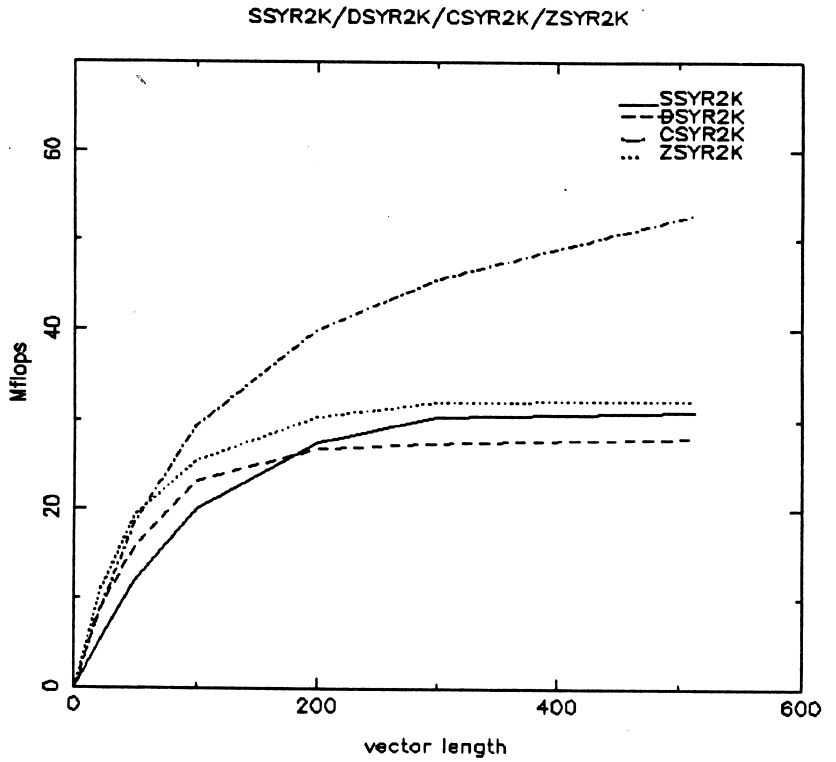

CHER2K/ZHER2K
CHER2K
ZHER2K
Mflops
0
20
40
60
0
200
400
600
vector length

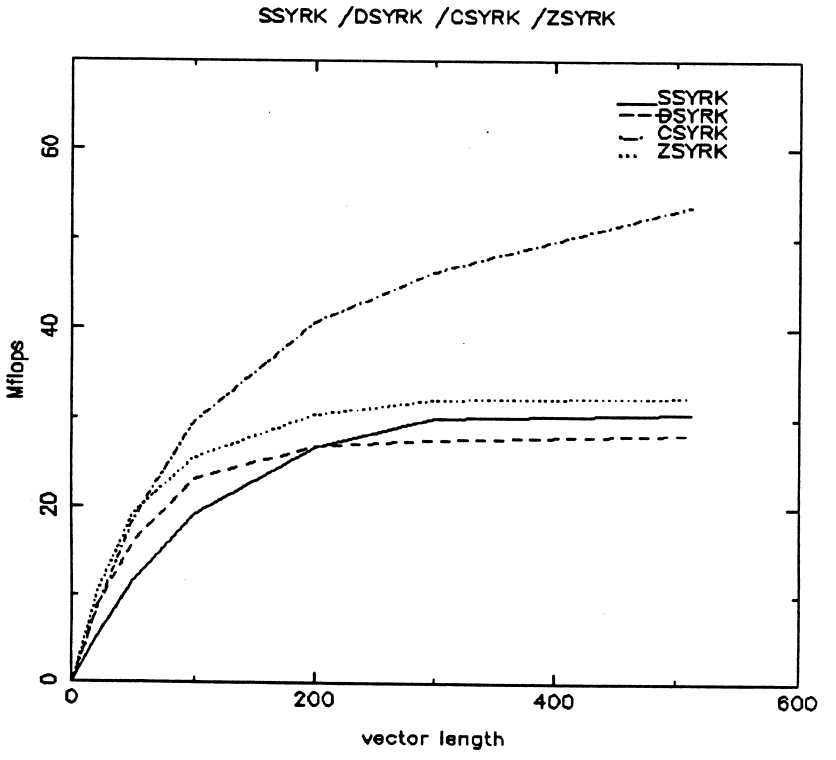
CHERK /ZHERK

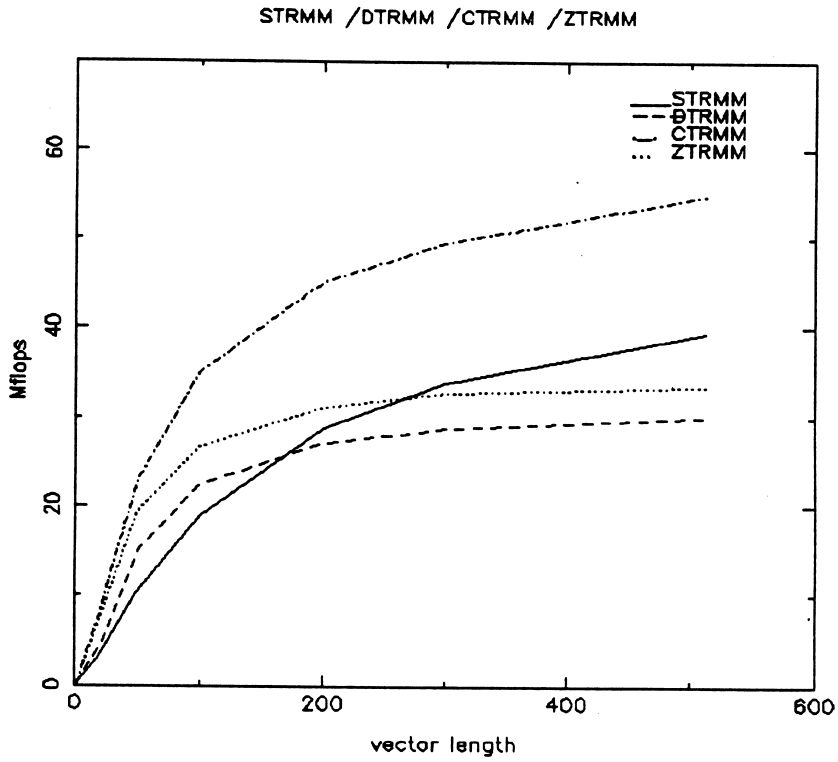


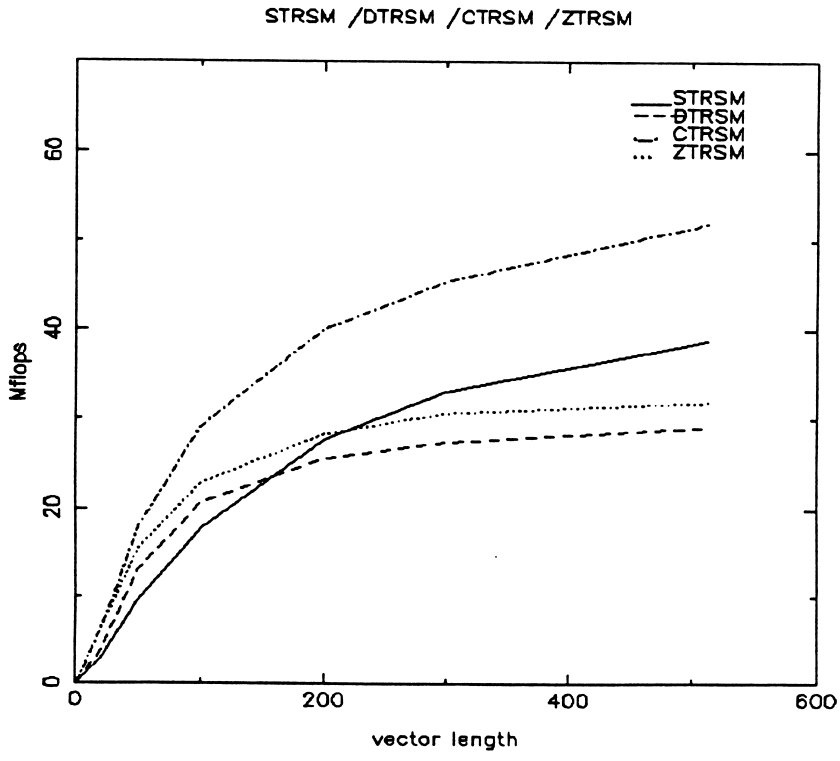
SSYMM /DSYMM /CSYMM /ZSYMM




SSYR2K/DSYR2K/CSYR2K/ZSYR2K
SSYR2K
DSYR2K
CSYR2K
ZSYR2K
Mflops
0
20
40
60
0
200
400
600
vector length


SSYRK /DSYRK /CSYRK /ZSYRK
SSYRK
DSYRK
CSYRK
ZSYRK
Mflops
0
20
40
60
0
200
400
600
vector length


STRMM /DTRMM /CTRMM /ZTRMM
STRMM
DTRMM
CTRMM
ZTRMM
Mflops
0
20
40
60
0
200
400
600
vector length


STRSM /DTRSM /CTRSM /ZTRSM
STRSM
DTRSM
CTRSM
ZTRSM
Mflops
0
20
40
60
0
200
400
600
vector length

FFT TEST CASES AND PERFORMANCE

The BLAS Math Library contains three routines for doing FFT's in place:1) Complex to complex (forward or inverse), 2) Real to complex (forward), 3) Complex to real (inverse).

Like most FFT software, the user first computes constants associated with a particular size (N) and then uses those constants to do a variable number (M) of FFT's. Performance (MFLOPS) is a function of M.

	Complex to Complex		Real to Complex Complex to Real	
N	M=1	M=100	M=1	M=100
32	3.0	16.0	2.2	5.7
64	4.0	24.0	4.2	24.0
128	4.9	34.4	5.2	40.7
256	5.7	39.4	6.2	60.2
512	6.5	42.6	7.1	69.8
1024	7.2	45.3	8.0	75.3

It is important to note that the time for constant generation was included in the data above. If the time for constant generation is not included, then results for M=1 and M=100 would be almost identical.

INDEX