Chapter 3

Dense Matrix Operations

This chapter describes the C* interface to the CMSSL dense matrix operations. One section is devoted to each of the following:

- inner product
- 2-norm
- outer product
- matrix vector multiplication
- vector matrix multiplication
- infinity norm
- matrix multiplication
- references

NOTE

The inner product, 2-norm, outer product, matrix vector multiplication, vector matrix multiplication, and matrix multiplication routines perform error checking only when the CMSSL safety mechanism is on. Therefore, we strongly recommend that you turn CMSSL safety on when testing new programs that call these routines.

Copyright © 1993 Thinking Machines Corporation

3.1 Inner Product

The *multiple-instance* inner product routines compute one or more instances of an inner product of two vectors. The inner product either overwrites the destination, is added to the destination, or is added to a second variable, with the results placed in the destination.

Given parallel variables **x, y, z**, and **u** containing multiple instances of the vectors x, y, z, and u, respectively, the multiple-instance inner product routines perform the operations listed below for each instance. The data types are abbreviated as follows:

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

Routine	Operation	Data Types
CMSSL_gen_inner_product	$z = z + x^{\mathrm{T}}y$	fdcz
CMSSL_gen_inner_product_noadd	$z = x^{\mathrm{T}}y$	fdcz
CMSSL_gen_inner_product_addto	$z = u + x^{\mathrm{T}}y$	fdcz
CMSSL_gen_inner_product_c1	$z = z + x^{\mathrm{H}}y$	cz
CMSSL_gen_inner_product_c1_noadd	$z = x^{\mathrm{H}}y$	cz
CMSSL_gen_inner_product_c1_addto	$z = u + x^{\mathrm{H}}y$	cz

Each *single-instance* (**CMSSL_gbl_**) inner product routine computes the global inner product over all axes of two source parallel variables. The inner product either overwrites the destination (which is a scalar variable), is added to the destination, or is added to a second variable, with the results placed in the destination.

Given parallel variables **x** and **y**, and scalars **a** and **b**, the single-instance inner product routines perform the operations listed below. (Note that the formulas below use the bold typewriter font for **x** and **y** because the single-instance routines compute the inner product over *all* axes of **x** and **y**. This is in contrast with the multiple-instance formulas above, which refer to single instances of the vectors x, y, z, and u embedded within **x, y, z**, and **u**, respectively.)

Copyright © 1993 Thinking Machines Corporation

Routine	Operation	Data Types
CMSSL_gbl_gen_inner_product	$\mathbf{a} = \mathbf{a} + \mathbf{x}^{T}\mathbf{y}$	**fdcz**
CMSSL_gbl_gen_inner_product_noadd	$\mathbf{a} = \mathbf{x}^{T}\mathbf{y}$	**fdcz**
CMSSL_gbl_gen_inner_product_addto	$\mathbf{a} = \mathbf{b} + \mathbf{x}^{T}\mathbf{y}$	**fdcz**
CMSSL_gbl_gen_inner_product_c1	$\mathbf{a} = \mathbf{a} + \mathbf{x}^{H}\mathbf{y}$	**cz**
CMSSL_gbl_gen_inner_product_c1_noadd	$\mathbf{a} = \mathbf{x}^{H}\mathbf{y}$	**cz**
CMSSL_gbl_gen_inner_product_c1_addto	$\mathbf{a} = \mathbf{b} + \mathbf{x}^{H}\mathbf{y}$	**cz**

Details are provided in the man page that follows.

Copyright © 1993 Thinking Machines Corporation

Inner Product

The multiple-instance (**CMSSL_gen_**) inner product routines compute one or more instances of an inner product of two vectors. The single-instance (**CMSSL_gbl_gen_**) inner product routines compute the global inner product over all axes of two source parallel variables.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_inner_product (
    fdcz:void *z, fdcz:void *x, fdcz:void *y,
    int x_vector_axis, int y_vector_axis);

int CMSSL_gen_inner_product_noadd (
    fdcz:void *z, fdcz:void *x, fdcz:void *y,
    int x_vector_axis, int y_vector_axis);

int CMSSL_gen_inner_product_addto (
    fdcz:void *z, fdcz:void *x, fdcz:void *y, fdcz:void *u,
    int x_vector_axis, int y_vector_axis);

int CMSSL_gen_inner_product_c1 (
    cz:void *z, cz:void *x, cz:void *y, int x_vector_axis,
    int y_vector_axis);

int CMSSL_gen_inner_product_c1_noadd (
    cz:void *z, cz:void *x, cz:void *y, int x_vector_axis,
    int y_vector_axis);

int CMSSL_gen_inner_product_c1_addto (
    cz:void *z, cz:void *x, cz:void *y, cz:void *u,
    int x_vector_axis, int y_vector_axis);

int CMSSL_gbl_gen_inner_product (
    fdcz *a, fdcz:void *x, fdcz:void *y);

int CMSSL_gbl_gen_inner_product_noadd (
    fdcz *a, fdcz:void *x, fdcz:void *y);

int CMSSL_gbl_gen_inner_product_addto (
    fdcz *a, fdcz:void *x, fdcz:void *y, fdcz b);

int CMSSL_gbl_gen_inner_product_c1 (
    cz *a, cz:void *x, cz:void *y);
```


Copyright © 1993 Thinking Machines Corporation

```
int CMSSL_gbl_gen_inner_product_c1_noadd (
    cz *a, cz:void *x, cz:void *y);

int CMSSL_gbl_gen_inner_product_c1_addto (
    cz *a, cz:void *x, cz:void *y, cz b);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

In a given inner product call, all variables listed as **fdcz** must have the same data type; all variables listed as **cz** must have the same data type. The **CMSSL_complex_t** and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

ARGUMENTS

z Pointer to a parallel variable of rank one less than that of **x** and **y**. The axes of **z** must match the instance axes of **x** and **y** in length and order of declaration. Thus, each pair of vectors in **x** and **y**, respectively, corresponds to a single destination value *z* in **z**.

x When you call one of the multiple-instance (**CMSSL_gen_**) routines, **x** must be a pointer to a parallel variable of rank ≥ 2, with at least one non-local instance axis. The variable **x** contains one or more instances of ***x***, the first vector in the pair of vectors whose inner product is to be computed. (For a single-instance problem, declare any instance axes to have length 1.) Axis **x_vector_axis** of **x** and axis **y_vector_axis** of **y** must have the same length. The remaining axes of **x** and **y** (the instance axes) must match in length and order of declaration.

When you call one of the single-instance (**CMSSL_gbl_gen_**) routines, **x** must be a pointer to a parallel variable of rank ≥ 1.

Copyright © 1993 Thinking Machines Corporation

y When you call one of the multiple-instance (**CMSSL_gen_**) routines, **y** must be a pointer to a parallel variable of the same rank as **x**, with at least one non-local instance axis. The variable **y** contains one or more instances of ***y***, the second vector in the pair of vectors whose inner product is to be computed. (For a single-instance problem, declare any instance axes to have length 1.) Axis **x_vector_axis** of **x** and axis **y_vector_axis** of **y** must have the same length. The remaining axes of **x** and **y** (the instance axes) must match in length and order of declaration.

When you call one of the single-instance (**CMSSL_gbl_gen_**) routines, **y** must be a pointer to a parallel variable of the same shape as **x**.

u Pointer to a parallel variable of rank one less than that of **x** and **y**, and the same shape as **z**. The axes of **u** must match the instance axes of **x** and **y** in length and order of declaration. Thus, each pair of vectors ***x*** and ***y*** in **x** and **y**, respectively, corresponds to a single source value ***u*** in **u**.

a Pointer to a scalar variable. Destination for the single-instance routines.

b Scalar variable. A source value for the single-instance routines.

x_vector_axis Scalar integer variable. Identifies the axis of **x** along which the vectors lie.

y_vector_axis Scalar integer variable. Identifies the axis of **y** along which the vectors lie.

RETURNED VALUE

Each routine returns an integer. Upon successful return, or if the CMSSL safety mechanism is turned off, 0 is returned.

If the CMSSL safety mechanism is turned on and an error occurs in a multiple-instance (**CMSSL_gen_**) call, the routine sets the return code as follows:

−4 The instance axes of **x** and **y** do not match in length and order of declaration.

−10 **z**, **x**, and **y** (and **u**, in **_addto** calls) are not all of the same data type and precision.

Copyright © 1993 Thinking Machines Corporation

−12 You called **CMSSL_gen_inner_product_c1**, **CMSSL_gen_inner_product_c1_noadd**, or **CMSSL_gen_inner_product_c1_addto**, but supplied data of a type other than **CMSSL_complex_t** or **CMSSL_dcomplex_t**.

−13 **x_vector_axis** or **y_vector_axis** is a bad axis number. Each must be non-negative and less than the rank of the corresponding parallel variable.

−22 **z** does not have rank one less than that of **x**.

−24 The axes of **z** do not match the instance axes of **x** in length and order of declaration.

If the CMSSL safety mechanism is turned on and an error occurs in a single-instance (**CMSSL_gbl_gen_**) call, the routine sets the return code as follows:

−2 **x** and **y** do not have the same rank.

−32 You called **CMSSL_gbl_gen_inner_product_c1**, **CMSSL_gbl_gen_inner_product_c1_noadd**, or **CMSSL_gbl_gen_inner_product_c1_addto**, but supplied data of a type other than **CMSSL_complex_t** or **CMSSL_dcomplex_t**.

DESCRIPTION

Multiple-Instance Routines. The multiple-instance inner product routines perform the operations listed below, for each instance.

Routine	Operation	Data Types
CMSSL_gen_inner_product	$z = z + x^{\mathrm{T}}y$	fdcz
CMSSL_gen_inner_product_noadd	$z = x^{\mathrm{T}}y$	fdcz
CMSSL_gen_inner_product_addto	$z = u + x^{\mathrm{T}}y$	fdcz
CMSSL_gen_inner_product_c1	$z = z + x^{\mathrm{H}}y$	cz
CMSSL_gen_inner_product_c1_noadd	$z = x^{\mathrm{H}}y$	cz
CMSSL_gen_inner_product_c1_addto	$z = u + x^{\mathrm{H}}y$	cz

These routines require the source parallel variables to be at least two-dimensional, with at least one non-local instance axis. (The reason for this restriction is that the destination must have rank one less than that of the sources, but also cannot be completely local. Thus, to compute the inner product of a *single* pair of vectors,

Copyright © 1993 Thinking Machines Corporation

you must either declare any instance axes to have length 1, or use the single-instance inner product routines.

Upon successful completion of **CMSSL_gen_inner_product** or **CMSSL_gen_inner_product_c1**, the inner product of each vector pair *x* and *y* in **x** and **y**, respectively, is added to the corresponding value in **z**.

Upon successful completion of **CMSSL_gen_inner_product_noadd** or **CMSSL_gen_inner_product_c1_noadd**, the inner product of each vector pair *x* and *y* in **x** and **y**, respectively, overwrites the corresponding value in **z**.

Upon successful completion of **CMSSL_gen_inner_product_addto** or **CMSSL_gen_inner_product_c1_addto**, the inner product of each vector pair *x* and *y* in **x** and **y**, respectively (added to the corresponding value in **u**) overwrites the corresponding value in **z**.

Single-Instance Routines. The single-instance inner product routines perform the operations listed below. (Note that the formulas below use the bold typewriter font for **x** and **y** because the single-instance routines compute the inner product over *all* axes of **x** and **y**. This is in contrast with the multiple-instance formulas above, which refer to single instances of the vectors ***x***, ***y***, *z*, and ***u*** embedded within **x, y, z**, and **u**, respectively.)

Routine	Operation	Data Types
CMSSL_gbl_gen_inner_product	$\mathbf{a} = \mathbf{a} + \mathbf{x}^{T}\mathbf{y}$	**fdcz**
CMSSL_gbl_gen_inner_product_noadd	$\mathbf{a} = \mathbf{x}^{T}\mathbf{y}$	**fdcz**
CMSSL_gbl_gen_inner_product_addto	$\mathbf{a} = \mathbf{b} + \mathbf{x}^{T}\mathbf{y}$	**fdcz**
CMSSL_gbl_gen_inner_product_c1	$\mathbf{a} = \mathbf{a} + \mathbf{x}^{H}\mathbf{y}$	**cz**
CMSSL_gbl_gen_inner_product_c1_noadd	$\mathbf{a} = \mathbf{x}^{H}\mathbf{y}$	**cz**
CMSSL_gbl_gen_inner_product_c1_addto	$\mathbf{a} = \mathbf{b} + \mathbf{x}^{H}\mathbf{y}$	**cz**

Upon successful completion of **CMSSL_gbl_gen_inner_product** or **CMSSL_gbl_gen_inner_product_c1**, the global inner product of **x** and **y** is added to **a**.

Upon successful completion of **CMSSL_gbl_gen_inner_product_noadd** or **CMSSL_gbl_gen_inner_product_c1_noadd**, the global inner product of **x** and **y** overwrites **a**.

Version 3.2 Beta, December 1993
Copyright © 1993 Thinking Machines Corporation

Upon successful completion of **CMSSL_gbl_gen_inner_product_addto** or **CMSSL_gbl_gen_inner_product_c1_addto**, the global inner product of **x** and **y** (added to **b**) overwrites **a**.

NOTES

Overlapping Variables. The parallel variables **x** and **y** may be the same variable; similarly, **z** and **u** may be the same variable.

Numerical Complexity: Multiple-Instance Routines. If the vectors contained in **x** and **y** have length q, then for I instances, the number of floating-point operations for real operands is

- $2qI$ for **CMSSL_gen_inner_product**, **CMSSL_gen_inner_product_c1**, **CMSSL_gen_inner_product_addto**, and **CMSSL_gen_inner_product_c1_addto**
- $(2q-1)I$ for **CMSSL_gen_inner_product_noadd** and **CMSSL_gen_inner_product_c1_noadd**

while the number of floating-point operations for complex operands is

- $8qI$ for **CMSSL_gen_inner_product**, **CMSSL_gen_inner_product_c1**, **CMSSL_gen_inner_product_addto**, and **CMSSL_gen_inner_product_c1_addto**
- $(8q-2)I$ for **CMSSL_gen_inner_product_noadd** and **CMSSL_gen_inner_product_c1_noadd**

Numerical Complexity: Single-Instance Routines. If the product of the axis lengths in each parallel variable (**x** and **y**) is Q, then the number of floating-point operations for real operands is

- $2Q$ for **CMSSL_gbl_gen_inner_product**, **CMSSL_gbl_gen_inner_product_c1**, **CMSSL_gbl_gen_inner_product_addto**, and **CMSSL_gbl_gen_inner_product_c1_addto**
- $2Q-1$ for **CMSSL_gbl_gen_inner_product_noadd** and **CMSSL_gbl_gen_inner_product_c1_noadd**

while the number of floating-point operations for complex operands is

- $8Q$ for **CMSSL_gbl_gen_inner_product**, **CMSSL_gbl_gen_inner_product_c1**, **CMSSL_gbl_gen_inner_product_addto**, and **CMSSL_gbl_gen_inner_product_c1_addto**

Copyright © 1993 Thinking Machines Corporation

- $8Q-2$ for **CMSSL_gbl_gen_inner_product_noadd** and **CMSSL_gbl_gen_inner_product_c1_noadd**

EXAMPLES

Sample C* code that uses the inner product routines can be found on-line in the subdirectory

```
inner-product/cstar/
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.2 2-Norm

The *multiple-instance* 2-norm routine, **CMSSL_gen_2_norm**, computes one or more instances of the 2-norm of a vector. Given a parallel variable **x** containing multiple instances of a vector ***x***, **CMSSL_gen_2_norm** performs the following operation for each instance:

Operation	Data Type of **x**
$z = (x^{T}x)^{1/2} = \|\|x\|\|_2$	**float** **double**
$z = (x^{H}x)^{1/2} = \|\|x\|\|_2$	**CMSSL_complex_t** **CMSSL_dcomplex_t**

The *single-instance* 2-norm routine, **CMSSL_gbl_gen_2_norm**, computes the global 2-norm of a parallel variable as defined below. (Note that the formulas below use the bold typewriter font for **x** because the single-instance routine computes the norm over *all* axes of **x**. This is in contrast with the multiple-instance formulas above, which refer to single instances of the vector ***x*** embedded within **x**.)

Operation	Data Type of **x**
$\mathbf{a} = (\mathbf{x}^{T}\mathbf{x})^{1/2} = \|\|\mathbf{x}\|\|_2$	**float** **double**
$\mathbf{a} = (\mathbf{x}^{H}\mathbf{x})^{1/2} = \|\|\mathbf{x}\|\|_2$	**CMSSL_complex_t** **CMSSL_dcomplex_t**

The norm is always a real (**float** or **double**) value. Details are provided in the man page that follows.

Copyright © 1993 Thinking Machines Corporation

2-Norm

The multiple-instance 2-norm routine, **CMSSL_gen_2_norm**, computes one or more instances of the 2-norm of a vector. The single-instance 2-norm routine, **CMSSL_gbl_gen_2_norm**, computes the global 2-norm of a parallel variable.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_2_norm (
    fdfd:void *z, fdcz:void *x, int x_vector_axis);

int CMSSL_gbl_gen_2_norm (
    fdfd *a, fdcz:void *x);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
fdfd **float** or **double**; same precision as **x**.

The **CMSSL_complex_t** and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

ARGUMENTS

z Pointer to a parallel variable of rank one less than that of **x**. The axes of **z** must match the instance axes of **x** in length and order of declaration. Thus, each vector ***x*** in **x** corresponds to a single destination value z in **z**.

a Pointer to a scalar variable. Destination for the single-instance 2-norm routine.

x When you call **CMSSL_gen_2_norm**, **x** must be a pointer to a parallel variable of rank ≥ 2, with at least one non-local instance axis. The variable **x** contains one or more instances of the vector ***x*** whose 2-norm you want to compute. (For a single-instance problem, declare any instance axes to have length 1.)

When you call **CMSSL_gbl_gen_2_norm**, **x** must be a pointer to a parallel variable of rank ≥ 1.

Copyright © 1993 Thinking Machines Corporation

x_vector_axis Scalar integer variable. Identifies the axis of **x** along which the vectors lie.

RETURNED VALUE

Each routine returns an integer. Upon successful return, or if the CMSSL safety mechanism is turned off, 0 is returned.

If the CMSSL safety mechanism is turned on and an error occurs in a **CMSSL_gen_2_norm** call, the routine sets the return code as follows:

−2 **x** does not have a rank of at least 2.

−13 **x_vector_axis** is a bad axis number. It must be non-negative and less than the rank of **x**.

−22 **z** does not have rank one less than that of **x**.

−24 The axes of **z** do not match the instance axes of **x** in length and order of declaration.

DESCRIPTION

Multiple-Instance Routine. For each instance, **CMSSL_gen_2_norm** performs the following operation:

Operation	Data Type of **x**
$z = (x^{\mathrm{T}}x)^{1/2} = \|\|x\|\|_2$	**float** **double**
$z = (x^{\mathrm{H}}x)^{1/2} = \|\|x\|\|_2$	**CMSSL_complex_t** **CMSSL_dcomplex_t**

The **CMSSL_gen_2_norm** routine requires the source parallel variable to be at least two-dimensional, with at least one non-local instance axis. (The reason for this restriction is that the destination must have rank one less than that of the source, but must also be a parallel variable — and therefore not completely local. Thus, to compute the 2-norm of a *single* vector, you must either declare any instance axes to have length 1, or use the single-instance 2-norm routine.

Upon successful completion of **CMSSL_gen_2_norm**, the 2-norm of each vector in **x** overwrites the corresponding value in **z**.

Single-Instance Routine. The **CMSSL_gbl_gen_2_norm** routine performs the operations listed below. (Note that the formulas below use the bold typewriter font for **x** because the single-instance routine computes the norm over *all* axes of **x**. This is in

Copyright © 1993 Thinking Machines Corporation

contrast with the multiple-instance formulas above, which refer to single instances of the vector ***x*** embedded within **x**.)

Operation	Data Type of **x**
$\mathbf{a} = (\mathbf{x}^{T}\mathbf{x})^{1/2} = \|\|\mathbf{x}\|\|_2$	**float** **double**
$\mathbf{a} = (\mathbf{x}^{H}\mathbf{x})^{1/2} = \|\|\mathbf{x}\|\|_2$	**CMSSL_complex_t** **CMSSL_dcomplex_t**

Upon successful completion of **CMSSL_gbl_gen_2_norm**, the global 2-norm of **x** overwrites **a**.

NOTES

Numerical Complexity: Multiple-Instance Routine. If the vectors contained in **x** have length q, then for I instances, the number of floating-point operations used by **CMSSL_gen_2_norm** is $[(2q-1)+8]I$ for real operands or $[(4q-1)+8]I$ for complex operands. (8 is the flop count for the square root operation.)

Numerical Complexity: Single-Instance Routine. If the product of the axis lengths of **x** is Q, then the number of floating-point operations used by **CMSSL_gbl_gen_2_norm** is $(2Q-1)+8$ for real operands or $(4Q-1)+8$ for complex operands. (8 is the flop count for the square root operation.)

EXAMPLES

Sample C* code that uses the 2-norm routines can be found on-line in the subdirectory

```
inner-product/cstar/
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.3 Outer Product

The outer product routines compute one or more instances of an outer product of two vectors. The result either overwrites the destination, is added to the destination, or is added to a second variable.

Given parallel variables **x**, **y**, **A**, and **B** containing multiple instances of the vectors x and y and the matrices A and B, respectively, the outer product routines perform the operations listed below for each instance. In these descriptions, y^{T} and y^{H} denote y transpose and y Hermitian, respectively. The data types are abbrviated as follows:

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

Routine	Operation	Data Types
CMSSL_gen_outer_product	$A = A + xy^{T}$	fdcz
CMSSL_gen_outer_product_noadd	$A = xy^{T}$	fdcz
CMSSL_gen_outer_product_addto	$A = B + xy^{T}$	fdcz
CMSSL_gen_outer_product_c2	$A = A + xy^{H}$	cz
CMSSL_gen_outer_product_c2_noadd	$A = xy^{H}$	cz
CMSSL_gen_outer_product_c2_addto	$A = B + xy^{H}$	cz

The man page following this section provides details.

Copyright © 1993 Thinking Machines Corporation

Outer Product

The routines described below compute one or multiple instances of an outer product of two vectors.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_outer_product (
    fdcz:void *A, fdcz:void *x, fdcz:void *y, int row_axis,
    int col_axis, int x_vector_axis, int y_vector_axis);

int CMSSL_gen_outer_product_noadd (
    fdcz:void *A, fdcz:void *x, fdcz:void *y, int row_axis,
    int col_axis, int x_vector_axis, int y_vector_axis);

int CMSSL_gen_outer_product_addto (
    fdcz:void *A, fdcz:void *x, fdcz:void *y, fdcz:void *B,
    int row_axis, int col_axis, int x_vector_axis,
    int y_vector_axis);

int CMSSL_gen_outer_product_c2 (
    cz:void *A, cz:void *x, cz:void *y, int row_axis,
    int col_axis, int x_vector_axis, int y_vector_axis);

int CMSSL_gen_outer_product_c2_noadd (
    cz:void *A, cz:void *x, cz:void *y, int row_axis,
    int col_axis, int x_vector_axis, int y_vector_axis);

int CMSSL_gen_outer_product_c2_addto (
    cz:void *A, cz:void *x, cz:void *y, cz:void *B,
    int row_axis, int col_axis, int x_vector_axis,
    int y_vector_axis);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

In a given outer product call, all variables listed as **fdcz** must have the same data type; all variables listed as **cz** must have the same data type. The **CMSSL_**

Copyright © 1993 Thinking Machines Corporation

complex_t and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

ARGUMENTS

A Pointer to a parallel variable of rank greater than or equal to 2. This variable contains one or more instances of the destination matrix, *A*, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). Upon completion, each matrix instance is overwritten by the result of the outer product call.

x Pointer to a parallel variable of rank one less than that of **A**. This variable contains one or more instances of the first source vector, *x*, embedded along axis **x_vector_axis**. Axis **x_vector_axis** of **x** must have the same length as axis **row_axis** of **A**. The remaining axes of **x** must match the instance axes of **A** in length and order of declaration. Thus, each vector in **x** corresponds to a matrix in **A**.

y Pointer to a parallel variable of rank one less than that of **A**. This variable contains one or more instances of the second source vector, *y*, embedded along axis **y_vector_axis**. Axis **y_vector_axis** of **y** must have the same length as axis **col_axis** of **A**. The remaining axes of **y** must match the instance axes of **A** in length and order of declaration. Thus, each vector in **y** corresponds to a matrix in **A**.

B Pointer to a parallel variable of the same shape as **A**. This variable contains one or more embedded matrices *B* defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). The remaining axes must match the instance axes of **A** in length and order of declaration. Thus, each matrix in **B** corresponds to a matrix in **A**. This argument is used only in the **CMSSL_gen_outer_product_addto** and **CMSSL_gen_outer_product_c2_addto** calls. These calls add each outer product to the corresponding matrix within **B** and place the result in the corresponding matrix within **A**. The contents of **B** are not changed by the operation (unless **B** and **A** are the same variable).

row_axis Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis of **A** and **B** that counts the rows of the embedded matrix or matrices.

Copyright © 1993 Thinking Machines Corporation

col_axis Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis of **A** and **B** that counts the columns of the embedded matrix or matrices.

x_vector_axis Scalar integer variable. Must be non-negative and less than the rank of **x**. The axis of **x** along which the elements of each embedded vector lie.

y_vector_axis Scalar integer variable. Must be non-negative and less than the rank of **y**. The axis of **y** along which the elements of each embedded vector lie.

RETURNED VALUE

Each routine returns an integer. Upon successful return, or if the CMSSL safety mechanism is turned off, 0 is returned. If the CMSSL safety mechanism is turned on and an error occurs, the routine sets the return code as follows:

−1 The rank of **A** < 2, or the rank of **x** or **y** is not equal to (rank of **A**)−1.

−2 The length of **x** along axis **x_vector_axis** is not equal to the number of rows in the matrices in **A**; or the length of **y** along axis **y_vector_axis** is not equal to the number of columns in the matrices in **A**.

−8 The instance axes of **x** and **y** do not match those of **A** in length and order of declaration.

−16 One or more of **row_axis**, **col_axis**, **x_vector_axis**, and **y_vector_axis** are bad axis numbers. Each must be non-negative and less than the rank of the corresponding parallel variable.

DESCRIPTION

For each instance, the outer product routines perform the operations listed below. In these descriptions, y^{T} and y^{H} denote y transpose and y Hermitian, respectively.

Routine	Operation	Data Types
CMSSL_gen_outer_product	$A = A + xy^{T}$	fdcz
CMSSL_gen_outer_product_noadd	$A = xy^{T}$	fdcz
CMSSL_gen_outer_product_addto	$A = B + xy^{T}$	fdcz
CMSSL_gen_outer_product_c2	$A = A + xy^{H}$	cz
CMSSL_gen_outer_product_c2_noadd	$A = xy^{H}$	cz

Copyright © 1993 Thinking Machines Corporation

CMSSL_gen_outer_product_c2_addto $A = B + xy^{H}$ **cz**

In elementwise notation, for each instance **CMSSL_gen_outer_product** computes

$$A(i,j) = A(i,j) + x(i) * y(j)$$

and **CMSSL_gen_outer_product_c2** computes

$$A(i,j) = A(i,j) + x(i) * \overline{y(j)}$$

where $\overline{y(j)}$ denotes the conjugate of $y(j)$.

NOTES

Distinct Variables. **A** must point to a variable that is distinct from the source variables pointed to by **x** and **y**. However, **x** and **y** can point to the same variable, and **B** and **A** can point to the same variable.

Numerical Stability. The algorithm for the outer product is numerically stable.

Numerical Complexity. If the matrices embedded in **A** and **B** have axis lengths (p x q), axis **x_vector_axis** of **x** has length p, and axis **y_vector_axis** of **y** has length q, then for I instances, the number of floating-point operations for real operands is

- $2pqI$ for **CMSSL_gen_outer_product** and **CMSSL_gen_outer_product_addto**
- pqI for **CMSSL_gen_outer_product_noadd**

while the number of floating-point operations for complex operands is

- $8pqI$ for **CMSSL_gen_outer_product** and **CMSSL_gen_outer_product_addto**
- $6pqI$ for **CMSSL_gen_outer_product_noadd**

Each conjugate routine performs the same number of floating-point operations as the corresponding non-conjugate routine.

EXAMPLES

Sample C* code that uses the outer product routines can be found on-line in the subdirectory **outer-product/cstar/** of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.4 Matrix Vector Multiplication

The matrix vector multiplication routines compute one or more matrix vector products. Given parallel variables **y**, **x**, **v**, and **A** containing multiple instances of the vectors y, x, and v and the matrix A, respectively, the matrix vector multiplication routines perform the operations listed below for each instance. In these descriptions, $\overline{A}$ denotes the conjugate of A. The data types are abbreviated as follows:

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

Routine	Operation	Data Types
CMSSL_gen_matrix_vector_mult	$y = y + Ax$	fdcz
CMSSL_gen_matrix_vector_mult_noadd	$y = Ax$	fdcz
CMSSL_gen_matrix_vector_mult_addto	$y = v + Ax$	fdcz
CMSSL_gen_matrix_vector_mult_c1	$y = y + \overline{A}x$	cz
CMSSL_gen_matrix_vector_mult_c1_noadd	$y = \overline{A}x$	cz
CMSSL_gen_matrix_vector_mult_c1_addto	$y = v + \overline{A}x$	cz

The man page following this section provides details.

Copyright © 1993 Thinking Machines Corporation

Matrix Vector Multiplication

The matrix vector multiplication routines compute one or more instances of a matrix vector product.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_matrix_vector_mult (
    fdcz:void *y, fdcz:void *A, fdcz:void *x,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);

int CMSSL_gen_matrix_vector_mult_noadd (
    fdcz:void *y, fdcz:void *A, fdcz:void *x,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);

int CMSSL_gen_matrix_vector_mult_addto (
    fdcz:void *y, fdcz:void *A, fdcz:void *x, fdcz:void *v,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);

int CMSSL_gen_matrix_vector_mult_c1 (
    cz:void *y, cz:void *A, cz:void *x, int y_vector_axis,
    int row_axis, int col_axis, int x_vector_axis);

int CMSSL_gen_matrix_vector_mult_c1_noadd (
    cz:void *y, cz:void *A, cz:void *x, int y_vector_axis,
    int row_axis, int col_axis, int x_vector_axis);

int CMSSL_gen_matrix_vector_mult_c1_addto (
    cz:void *y, cz:void *A, cz:void *x, cz:void *v,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

In a given matrix vector product call, all variables listed as **fdcz** must have the same data type; all variables listed as **cz** must have the same data type. The

Copyright © 1993 Thinking Machines Corporation

CMSSL_complex_t and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

ARGUMENTS

y Pointer to a parallel variable of rank greater than or equal to 1. This variable contains one or more instances of the destination vector *y*, embedded along axis **y_vector_axis**. Axis **y_vector_axis** of **y** must have the same length as axis **row_axis** of **A**. Upon completion, each vector instance is overwritten by the result of the matrix vector multplication call.

A Pointer to a parallel variable of rank one greater than that of **y**. This variable contains one or more instances of the matrix *A*, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). The remaining axes must match the instance axes of **y** in length and order of declaration. Thus, each matrix in **A** corresponds to a vector in **y**. The contents of **A** are not changed during execution.

x Pointer to a parallel variable of the same rank as **y**. This variable contains one or more instances of *x*, the vector that is to be multiplied by the matrix *A*, embedded along axis **x_vector_axis**. Axis **x_vector_axis** of **x** must have the same length as axis **col_axis** of **A**. The remaining axes of **x** must match the instance axes of **y** in length and order of declaration. Thus, each vector in **x** corresponds to a vector in **y**. The contents of **x** are not changed during execution.

v Pointer to a parallel variable of the same rank, shape as **y**. This argument is used only in the **CMSSL_gen_matrix_vector_mult_addto** and **CMSSL_gen_matrix_vector_mult_c1_addto** calls. It contains one or more instances of the vector *v* that is to be added to the matrix vector product, embedded along axis **y_vector_axis**. The contents of **v** are not changed during execution, unless **v** is the same variable as **y**.

y_vector_axis Scalar integer variable. Must be non-negative and less than the rank of **y**. The axis of **y** and **v** along which the elements of the embedded vectors lie.

row_axis Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis of **A** that counts the rows of the embedded matrix or matrices.

Copyright © 1993 Thinking Machines Corporation

col_axis Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis of **A** that counts the columns of the embedded matrix or matrices.

x_vector_axis Scalar integer variable. Must be non-negative and less than the rank of **x**. The axis of **x** along which the elements of the embedded vectors lie.

RETURNED VALUE

Each routine returns an integer. Upon successful return, or if the CMSSL safety mechanism is turned off, 0 is returned. If the CMSSL safety mechanism is turned on and an error occurs, the routine sets the return code as follows:

−1 Rank(**x**) ≠ rank(**y**) ≠ rank(**A**) −1.

−2 Axis **row_axis** of **A** and axis **y_vector_axis** of **y** do not have the same length, or axis **col_axis** of **A** and axis **x_vector_axis** of **x** do not have the same length.

−8 Instance axes of the input parallel variables do not match in length and order of declaration; or **y** and **v** do not have the same shape.

DESCRIPTION

For each instance, the matrix vector multiplication routines perform the operations listed below. In these descriptions, $\overline{A}$ denotes the conjugate of A.

Routine	Operation	Data Types
CMSSL_gen_matrix_vector_mult	$y = y + Ax$	fdcz
CMSSL_gen_matrix_vector_mult_noadd	$y = Ax$	fdcz
CMSSL_gen_matrix_vector_mult_addto	$y = v + Ax$	fdcz
CMSSL_gen_matrix_vector_mult_c1	$y = y + \overline{A}x$	cz
CMSSL_gen_matrix_vector_mult_c1_noadd	$y = \overline{A}x$	cz
CMSSL_gen_matrix_vector_mult_c1_addto	$y = v + \overline{A}x$	cz

NOTES

Distinct Variables. The variables pointed to by **y**, **A**, and **x** must be distinct. However, **v** and **y** can point to the same variable.

Numerical Stability. The algorithm is numerically stable.

Copyright © 1993 Thinking Machines Corporation

Numerical Complexity. If the matrices embedded in **A** have axis lengths ($p \times q$), axis **x_vector_axis** of **x** has length q, and axis **y_vector_axis** of **y** has length p, then for I instances, the number of floating-point operations performed is shown below.

	Real Operands	Complex Operands
CMSSL_gen_matrix_vector_mult	$2pqI$	$8pqI$
CMSSL_gen_matrix_vector_mult_addto	$2pqI$	$8pqI$
CMSSL_gen_matrix_vector_mult_noadd	$(2pq - p)\ I$	$(8pq - 2p)\ I$

Each conjugate routine performs the same number of floating-point operations as the corresponding non-conjugate routine.

EXAMPLES

Sample C* code that uses the matrix vector multiplication routines can be found on-line in the subdirectory

```
matrix-vector/cstar/
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.5 Vector Matrix Multiplication

The vector matrix multiplication routines compute one or more vector matrix products. Given parallel variables **y**, **x**, **v**, and **A** containing multiple instances of the vectors y, x, and v and the matrix A, respectively, the vector matrix multiplication routines perform the operations listed below for each instance. In these descriptions, $\overline{A}$ denotes the conjugate of A. The data types are abbreviated as follows:

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

Routine	Operation	Data Types
CMSSL_gen_vector_matrix_mult	$y^T = y^T + x^T A$	fdcz
CMSSL_gen_vector_matrix_mult_noadd	$y^T = x^T A$	fdcz
CMSSL_gen_vector_matrix_mult_addto	$y^T = v^T + x^T A$	fdcz
CMSSL_gen_vector_matrix_mult_c2	$y^T = y^T + x^T \overline{A}$	cz
CMSSL_gen_vector_matrix_mult_c2_noadd	$y^T = x^T \overline{A}$	cz
CMSSL_gen_vector_matrix_mult_c2_addto	$y^T = v^T + x^T \overline{A}$	cz

The man page following this section provides details.

Copyright © 1993 Thinking Machines Corporation

Vector Matrix Multiplication

The vector matrix multiplication routines compute one or more instances of a vector matrix product.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_vector_matrix_mult(
    fdcz:void *y, fdcz:void *A, fdcz:void *x,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);

int CMSSL_gen_vector_matrix_mult_noadd(
    fdcz:void *y, fdcz:void *A, fdcz:void *x,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);

int CMSSL_gen_vector_matrix_mult_addto(
    fdcz:void *y, fdcz:void *A, fdcz:void *x, fdcz:void *v,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);

int CMSSL_gen_vector_matrix_mult_c2(
    cz:void *y, cz:void *A, cz:void *x, int y_vector_axis,
    int row_axis, int col_axis, int x_vector_axis);

int CMSSL_gen_vector_matrix_mult_c2_noadd(
    cz:void *y, cz:void *A, cz:void *x, int y_vector_axis,
    int row_axis, int col_axis, int x_vector_axis);

int CMSSL_gen_vector_matrix_mult_c2_addto(
    cz:void *y, cz:void *A, cz:void *x, cz:void *v,
    int y_vector_axis, int row_axis, int col_axis,
    int x_vector_axis);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

In a given vector matrix product call, all variables listed as **fdcz** must have the same data type; all variables listed as **cz** must have the same data type. The

Copyright © 1993 Thinking Machines Corporation

CMSSL_complex_t and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

ARGUMENTS

y Pointer to a parallel variable of rank greater than or equal to 1. This variable contains one or more instances of the destination vector *y*, embedded along axis **y_vector_axis**. Axis **y_vector_axis** of **y** must have the same length as axis **col_axis** of **A**. Upon successful completion, each vector instance is overwritten by the result of the vector matrix multplication call.

A Pointer to a parallel variable of rank one greater than that of **y**. This variable contains one or more instances of the matrix *A*, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). The remaining axes must match the instance axes of **y** in length and order of declaration. Thus, each matrix in **A** corresponds to a vector in **y**. The contents of **A** are not changed during execution.

x Pointer to a parallel variable of the same rank as **y**. This variable contains one or more instances of the vector *x* that is to be multiplied by the matrix *A*, embedded along axis **x_vector_axis**. Axis **x_vector_axis** of **x** must have the same length as axis **row_axis** of **A**. The remaining axes of **x** must match the instance axes of **y** in length and order of declaration. Thus, each vector in **x** corresponds to a vector in **y**. The contents of **x** are not changed during execution.

v Pointer to a parallel variable of the same shape as **y**. This argument is used only in the **CMSSL_gen_vector_matrix_mult_addto** and **CMSSL_gen_vector_matrix_mult_c2_addto** calls. It contains one or more instances of the vector *v* that is to be added to the vector matrix product, embedded along axis **y_vector_axis**. The contents of **v** are not changed during execution, unless **v** is the same variable as **y**.

y_vector_axis Scalar integer variable. Must be non-negative and less than the rank of **y**. The axis of **y** and **v** along which the elements of the embedded vectors lie.

row_axis Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis of **A** that counts the rows of the embedded matrix or matrices.

Copyright © 1993 Thinking Machines Corporation

col_axis Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis of **A** that counts the columns of the embedded matrix or matrices.

x_vector_axis Scalar integer variable. Must be non-negative and less than the rank of **x**. The axis of **x** along which the elements of each vector lie.

RETURNED VALUE

Each routine returns an integer. Upon successful return, or if the CMSSL safety mechanism is turned off, 0 is returned. If the CMSSL safety mechanism is turned on and an error occurs, the routine sets the return code as follows:

−1 Rank(**x**) ≠ rank(**y**) ≠ rank(**A**) −1.

−2 Axis **col_axis** of **A** and axis **y_vector_axis** of **y** do not have the same length, or axis **row_axis** of **A** and axis **x_vector_axis** of **x** do not have the same length.

−8 Instance axes of the input parallel variables do not match in length and order of declaration; or **y** and **v** do not have the same shape.

DESCRIPTION

For each instance, the vector matrix multiplication routines perform the operations listed below. In these descriptions, $\overline{A}$ denotes the conjugate of A.

Routine	Operation	Data Types
CMSSL_gen_vector_matrix_mult	$y^T = y^T + x^T A$	fdcz
CMSSL_gen_vector_matrix_mult_noadd	$y^T = x^T A$	fdcz
CMSSL_gen_vector_matrix_mult_addto	$y^T = v^T + x^T A$	fdcz
CMSSL_gen_vector_matrix_mult_c2	$y^T = y^T + x^T \overline{A}$	cz
CMSSL_gen_vector_matrix_mult_c2_noadd	$y^T = x^T \overline{A}$	cz
CMSSL_gen_vector_matrix_mult_c2_addto	$y^T = v^T + x^T \overline{A}$	cz

NOTES

Distinct Variables. The variables pointed to by **y**, **A**, and **x** must be distinct. However, **v** and **y** can point to the same variable.

Numerical Stability. The algorithm is numerically stable.

Copyright © 1993 Thinking Machines Corporation

Numerical Complexity. If the matrices embedded in **A** have axis lengths (p x q), axis **x_vector_axis** of **x** has length q, and axis **y_vector_axis** of **y** has length p, then for I instances, the number of floating-point operations performed is shown below.

	Real Operands	Complex Operands
gen_vector_matrix_mult	$2pqI$	$8pqI$
gen_vector_matrix_mult_addto	$2pqI$	$8pqI$
gen_vector_matrix_mult_noadd	$(2pq - q)\ I$	$(8pq - 2q)\ I$

Each conjugate routine performs the same number of floating-point operations as the corresponding non-conjugate routine.

EXAMPLES

Sample C* code that uses the vector matrix multiplication routines can be found on-line in the subdirectory

```
vector-matrix/cstar/
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.6 Infinity Norm

Given a parallel variable **A** containing one or more matrices A, the **CMSSL_gen_infinity_norm** routine computes the infinity norm of each matrix A. Details are provided in the man page that follows.

Version 3.2 Beta, December 1993
Copyright © 1993 Thinking Machines Corporation

Infinity Norm

Given one or more matrices embedded in a parallel variable, the **CMSSL_gen_infinity_norm** routine computes the infinity norms of the matrices.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_infinity_norm (
    fdfd:void *A_norm, fdcz:void *A, int m, int n,
    int row_axis, int col_axis);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
fdfd **float** or **double**; same precision as **A**.

The **CMSSL_complex_t** and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

ARGUMENTS

A_norm Pointer to a parallel variable with the same rank as **A**. Axes **row_axis** and **col_axis** must have length 1.

Upon successful completion of **CMSSL_gen_infinity_norm**, the infinity norm of each matrix A within **A** is placed in the corresponding position of **A_norm**. For example, if **A** has dimensions [16][16][4][128], with **row_axis** = 1 and **col_axis** = 2, then upon completion, [r][0][0][s]**A_norm** contains the infinity norm of the matrix consisting of [r][.][.][s].

A Pointer to a parallel variable of rank $\geq$ 2.

When you call **CMSSL_gen_infinity_norm**, **A** must contain one or more embedded matrices A whose infinity norms you want to compute. Each matrix A is assumed to be dense with dimensions **m** x **n**. The axis identified by **row_axis** must count the rows of the embedded matrices; the axis identified by **col_axis** must count the columns of the matrices. Axes **row_axis** and **col_axis** may have lengths greater than **m** and **n**, respectively; that is, each instance of A may be contained in the upper left-hand **m** x **n** elements of a larger matrix within **A**.

Copyright © 1993 Thinking Machines Corporation

m	Scalar integer variable. The number of rows in each matrix embedded in **A**.
n	Scalar integer variable. The number of columns in each matrix embedded in **A**.
row_axis	Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis that counts the rows of the matrices A embedded in **A**.
col_axis	Scalar integer variable. Must be non-negative and less than the rank of **A**. The axis that counts the columns of the matrices A embedded in **A**.

RETURNED VALUE

The infinity norm routine returns 0 on successful return, or one of the following error codes:

−1	**m** is invalid.
−2	**n** is invalid.
−8	The rank of **A_norm** is not equal to the rank of **A**.
−64	**row_axis** or **col_axis** is invalid.

DESCRIPTION

Given one or more matrices A embedded in a parallel variable **A**, the **CMSSL_gen_infinity_norm** routine computes the infinity norm of each A.

The infinity norm of a matrix A, denoted here by $||A||_\infty$, is defined by

$$\|A\|_\infty = \max_{\|x\|_\infty = 1} \|Ax\|_\infty$$

where the infinity norm of a vector, $||x||_\infty$, is defined as the maximum of the absolute values of the vector components:

Version 3.2 Beta, December 1993
Copyright © 1993 Thinking Machines Corporation

$$\| \boldsymbol{x} \|_\infty = \max_i | x_i |$$

The infinity-norm condition number of a matrix $\boldsymbol{A}$ is equal to the product of $||\boldsymbol{A}||_\infty$ and $||\boldsymbol{A}^{-1}||_\infty$.

EXAMPLES

Sample C* code that uses the **CMSSL_gen_infinity_norm** routine can be found on-line in the subdirectory

```
infinity-norm/cstar/
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.7 Matrix Multiplication

The matrix multiplication routines compute one or more matrix products. Given parallel variables **A**, **B**, **C**, and **D** containing multiple instances of the matrices A, B, C, and D, respectively, the matrix multiplication routines perform the operations listed below for each instance. In these descriptions, A^{T} and A^{H} denote A transpose and A Hermitian, respectively. The data types are abbreviated as follows:

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

Routine	Operation	Data Types
CMSSL_gen_matrix_mult	$C = C + AB$	fdcz
CMSSL_gen_matrix_mult_noadd	$C = AB$	fdcz
CMSSL_gen_matrix_mult_addto	$C = D + AB$	fdcz
CMSSL_gen_matrix_mult_t1	$C = C + A^{\mathrm{T}}B$	fdcz
CMSSL_gen_matrix_mult_t1_noadd	$C = A^{\mathrm{T}}B$	fdcz
CMSSL_gen_matrix_mult_t1_addto	$C = D + A^{\mathrm{T}}B$	fdcz
CMSSL_gen_matrix_mult_h1	$C = C + A^{\mathrm{H}}B$	cz
CMSSL_gen_matrix_mult_h1_noadd	$C = A^{\mathrm{H}}B$	cz
CMSSL_gen_matrix_mult_h1_addto	$C = D + A^{\mathrm{H}}B$	cz
CMSSL_gen_matrix_mult_t2	$C = C + AB^{\mathrm{T}}$	fdcz
CMSSL_gen_matrix_mult_t2_noadd	$C = AB^{\mathrm{T}}$	fdcz
CMSSL_gen_matrix_mult_t2_addto	$C = D + AB^{\mathrm{T}}$	fdcz
CMSSL_gen_matrix_mult_h2	$C = C + AB^{\mathrm{H}}$	cz
CMSSL_gen_matrix_mult_h2_noadd	$C = AB^{\mathrm{H}}$	cz
CMSSL_gen_matrix_mult_h2_addto	$C = D + AB^{\mathrm{H}}$	cz
CMSSL_gen_matrix_mult_t1_t2	$C = C + A^{\mathrm{T}}B^{\mathrm{T}}$	fdcz

Copyright © 1993 Thinking Machines Corporation

CMSSL_gen_matrix_mult_t1_t2_noadd	$C = A^{\mathrm{T}}B^{\mathrm{T}}$	`fdcz`
CMSSL_gen_matrix_mult_t1_t2_addto	$C = D + A^{\mathrm{T}}B^{\mathrm{T}}$	`fdcz`

The algorithm used depends on the axis lengths of the parallel variables supplied. The man page following this section provides details about this routine.

Copyright © 1993 Thinking Machines Corporation

Matrix Multiplication

The matrix multiplication routines compute one or more matrix products.

SYNTAX

```
#include <cm/cmssl.h>

int CMSSL_gen_matrix_mult (
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_noadd (
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_addto (
    fdcz:void *C, fdcz:void *A, fdcz:void *B, fdcz:void *D,
    int row_axis, int col_axis);

int CMSSL_gen_matrix_mult_t1 (
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_t1_noadd (
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_t1_addto (
    fdcz:void *C, fdcz:void *A, fdcz:void *B, fdcz:void *D,
    int row_axis, int col_axis);

int CMSSL_gen_matrix_mult_h1 (
    cz:void *C, cz:void *A, cz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_h1_noadd (
    cz:void *C, cz:void *A, cz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_h1_addto (
    cz:void *C, cz:void *A, cz:void *B, cz:void *D,
    int row_axis, int col_axis);
```


Copyright © 1993 Thinking Machines Corporation

```
int CMSSL_gen_matrix_mult_t2(
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_t2_noadd(
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_t2_addto(
    fdcz:void *C, fdcz:void *A, fdcz:void *B, fdcz:void *D,
    int row_axis, int col_axis);

int CMSSL_gen_matrix_mult_h2(
    cz:void *C, cz:void *A, cz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_h2_noadd(
    cz:void *C, cz:void *A, cz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_h2_addto(
    cz:void *C, cz:void *A, cz:void *B, cz:void *D,
    int row_axis, int col_axis);

int CMSSL_gen_matrix_mult_t1_t2(
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_t1_t2_noadd(
    fdcz:void *C, fdcz:void *A, fdcz:void *B, int row_axis,
    int col_axis);

int CMSSL_gen_matrix_mult_t1_t2_addto(
    fdcz:void *C, fdcz:void *A, fdcz:void *B, fdcz:void *D,
    int row_axis, int col_axis);
```

OVERLOADED DATA TYPES

fdcz **float**, **double**, **CMSSL_complex_t**, or **CMSSL_dcomplex_t**
cz **CMSSL_complex_t** or **CMSSL_dcomplex_t**

In a given matrix product call, all variables listed as **fdcz** must have the same data type; all variables listed as **cz** must have the same data type. The **CMSSL_complex_t** and **CMSSL_dcomplex_t** data types are defined in the header file **<cm/cmssl.h>**.

Copyright © 1993 Thinking Machines Corporation

ARGUMENTS

C Pointer to a parallel variable of rank greater than or equal to 2. This variable contains one or more instances of the destination matrix ***C***, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). Axis **row_axis** of **C** must have the same length as axis **row_axis** of **A**. Axis **col_axis** of **C** must have the same length as axis **col_axis** of **B**.

Upon successful completion, each matrix instance within **C** is overwritten by the result of the matrix multiplication call.

A Pointer to a parallel variable of the same rank as **C**. This variable contains one or more instances of the left-hand factor matrix ***A***, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). Axis **col_axis** of **A** must have the same length as axis **row_axis** of **B**. The contents of **A** are not changed during execution.

B Pointer to a parallel variable of the same rank as **C**. This variable contains one or more instances of the right-hand factor matrix ***B***, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). The contents of **B** are not changed during execution.

D Pointer to a parallel variable of the same shape as **C**. This argument is used only in the calls whose names end in "**_addto**." It contains one or more instances of the matrix ***D*** that is to be added to the matrix product, defined by axes **row_axis** (which counts the rows) and **col_axis** (which counts the columns). The contents of **D** are not changed during execution, unless **D** and **C** are the same variable.

row_axis Scalar integer variable. Must be non-negative and less than the rank of **C**. The axis of **C**, **A**, **B**, and **D** that counts the rows of the embedded matrix or matrices.

col_axis Scalar integer variable Must be non-negative and less than the rank of **C**. The axis of **C**, **A**, **B**, and **D** that counts the columns of the embedded matrix or matrices.

Copyright © 1993 Thinking Machines Corporation

RETURNED VALUE

Each routine returns an integer. Upon successful return, or if the CMSSL safety mechanism is turned off, 0 is returned. If the CMSSL safety mechanism is turned on and an error occurs, the routine sets the return code as follows:

−1 Ranks of provided parallel variables are different or are not at least 2.

−2 Non-conforming **A** and **B** axis lengths.

−4 Non-conforming **A** and **C** axis lengths.

−8 Non-conforming **B** and **C** axis lengths.

−16 The instance axes of **A**, **B**, and **C** do not match in length and order of declaration.

−32 **row_axis** and/or **col_axis** is a bad axis number. Each must be non-negative and less than the rank of **C**.

DESCRIPTION

For each instance, the matrix multiplication routines perform the operations listed below. In these descriptions, A^{T} and A^{H} denote A transpose and A Hermitian, respectively.

Routine	Operation	Data Types
CMSSL_gen_matrix_mult	$C = C + AB$	fdcz
CMSSL_gen_matrix_mult_noadd	$C = AB$	fdcz
CMSSL_gen_matrix_mult_addto	$C = D + AB$	fdcz
CMSSL_gen_matrix_mult_t1	$C = C + A^{\mathrm{T}}B$	fdcz
CMSSL_gen_matrix_mult_t1_noadd	$C = A^{\mathrm{T}}B$	fdcz
CMSSL_gen_matrix_mult_t1_addto	$C = D + A^{\mathrm{T}}B$	fdcz
CMSSL_gen_matrix_mult_h1	$C = C + A^{\mathrm{H}}B$	cz
CMSSL_gen_matrix_mult_h1_noadd	$C = A^{\mathrm{H}}B$	cz
CMSSL_gen_matrix_mult_h1_addto	$C = D + A^{\mathrm{H}}B$	cz
CMSSL_gen_matrix_mult_t2	$C = C + AB^{\mathrm{T}}$	fdcz

Copyright © 1993 Thinking Machines Corporation

CMSSL_gen_matrix_mult_t2_noadd	$C = AB^{\mathrm{T}}$	**fdcz**
CMSSL_gen_matrix_mult_t2_addto	$C = D + AB^{\mathrm{T}}$	**fdcz**
CMSSL_gen_matrix_mult_h2	$C = C + AB^{\mathrm{H}}$	**cz**
CMSSL_gen_matrix_mult_h2_noadd	$C = AB^{\mathrm{H}}$	**cz**
CMSSL_gen_matrix_mult_h2_addto	$C = D + AB^{\mathrm{H}}$	**cz**
CMSSL_gen_matrix_mult_t1_t2	$C = C + A^{\mathrm{T}}B^{\mathrm{T}}$	**fdcz**
CMSSL_gen_matrix_mult_t1_t2_noadd	$C = A^{\mathrm{T}}B^{\mathrm{T}}$	**fdcz**
CMSSL_gen_matrix_mult_t1_t2_addto	$C = D + A^{\mathrm{T}}B^{\mathrm{T}}$	**fdcz**

The algorithm used depends on the axis lengths of the variables supplied.

For calls that do not transpose the matrices, the variables conform correctly with the following axis lengths for **row_axis** and **col_axis**:

Variable	**row_axis** length	**col_axis** length
A	p	q
B	q	r
C	p	r
D	p	r

For calls that transpose the matrix A, the variables conform correctly with the following axis lengths for **row_axis** and **col_axis**:

Variable	**row_axis** length	**col_axis** length
A	q	p
B	q	r
C	p	r
D	p	r

For calls that transpose the matrix B, the variables conform correctly with the following axis lengths for **row_axis** and **col_axis**:

Version 3.2 Beta, December 1993
Copyright © 1993 Thinking Machines Corporation

Variable	`row_axis` length	`col_axis` length
`A`	p	q
`B`	r	q
`C`	p	r
`D`	p	r

For calls that transpose both A and B, the variables conform correctly with the following axis lengths for `row_axis` and `col_axis`:

Variable	`row_axis` length	`col_axis` length
`A`	q	p
`B`	r	q
`C`	p	r
`D`	p	r

NOTES

Distinct Variables. All input parallel variables must be distinct, except that `C` and `D` can point to the same variable.

Numerical Stability. The algorithm is numerically stable.

Numerical Complexity. If the matrices embedded in `A` have the axis lengths listed in the Description section, then for I instances, the number of floating-point operations performed is shown below.

	Real Operands	Complex Operands
gen_matrix_mult	$2pqrI$	$8pqrI$
gen_matrix_mult_addto	$2pqrI$	$8pqrI$
gen_matrix_mult_noadd	$(2pqr - pr)\ I$	$(8pqr - 2pr)\ I$

Each Hermitian routine performs the same number of floating-point operations as the corresponding non-Hermitian routine.

EXAMPLES

Sample C* code that uses the matrix multiplication routines can be found on-line in the subdirectory

Copyright © 1993 Thinking Machines Corporation

```
matrix-multiply/cstar/
```

of a CMSSL examples directory whose location is site-specific.

Copyright © 1993 Thinking Machines Corporation

3.8 References

For more information about the basic linear routines for dense matrices, see the following references:

1. Dongarra, J. J., J. Du Croz, S. Hammarling, and R. J. Hanson. *An Extended Set of Fortran Basic Linear Algebra Subprograms*. Argonne National Laboratories, Mathematics and Computer Science Division, Technical Memorandum 41, November 1986.

2. Dongarra, J. J., J. Du Croz, I. Duff, and S. Hammerling. *A Set of Level 3 Basic Linear Algebra Subprograms*. Argonne National Laboratories, Mathematics and Computer Science Division, Reprint No. 1, August 1988.

3. Dongarra, J. J., J. Du Croz, I. Duff, and S. Hammerling. *A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs*. Argonne National Laboratories, Mathematics and Computer Science Division, Reprint No. 2, August 1988.

4. Golub, G. H., and Van Loan, C. F. *Matrix Computations*. 2d ed. Baltimore: Johns Hopkins University Press, 1989; or any basic linear algebra text.

5. Johnsson, S. L. Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures. *Journal of Parallel and Distributed Computing* **4** (1987): 133−72.

6. Johnsson, S. L., T. Harris, and K. K. Mathur. Matrix Multiplication on the Connection Machine. In *Proceedings of Supercomputing '89*, ACM Press, New York, 1989. Pp. 326−32.

7. Cannon, L. E. *A Cellular Computer to Implement the Kalman Filter Algorithm*. Ph.D. diss., Montana State University, 1969.

8. Mathur, K. K. and S. L. Johnsson. Multiplication of Matrices of Arbitrary Shape on a Data Parallel Computer. Thinking Machines Corporation Technical Report TR−216, 1992.

9. Johnsson, S. L. and L. Ortiz. Local Basic Linear Algebra Subroutines (LBLAS) for Distributed Memory Architectures and Languages with Array Syntax. *Int. J. Supercomputer App*. **6**, no. 4 (1992): 322−50.

For more information specifically about the infinity norm, see the following references:

Copyright © 1993 Thinking Machines Corporation

10. Hager, W. W. Condition Estimates. *SIAM J. Sci. Stat. Comput.* **5** (1984): 311–16.

11. Higham, N. J. Experience with a Matrix Norm Estimator. *SIAM J. Sci. Stat. Comput.* **11**, no. 4 (1990): 804–9.

12. Higham, N. J. FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation. (Algorith 674) *ACM Trans. Math. Soft.* **14** (1988): 381–96.

Copyright © 1993 Thinking Machines Corporation