Chapter 3

Dense Matrix Operations

This chapter describes the CM Fortran interface to the CMSSL dense matrix operations, which provide a subset of Basic Linear Algebra Subroutines (BLAS) functionality. 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
- matrix multiplication with external storage
- 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 © 1994 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 CM array, is added to the destination CM array, or is added to a second CM array (with the results placed in the destination CM array).

Given CM arrays ***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.

Routine	Operation	Data Types
gen_inner_product	$z = z + x^{\mathrm{T}}y$	real or complex
gen_inner_product_noadd	$z = x^{\mathrm{T}}y$	real or complex
gen_inner_product_addto	$z = u + x^{\mathrm{T}}y$	real or complex
gen_inner_product_c1	$z = z + x^{\mathrm{H}}y$	complex only
gen_inner_product_c1_noadd	$z = x^{\mathrm{H}}y$	complex only
gen_inner_product_c1_addto	$z = u + x^{\mathrm{H}}y$	complex only

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

Given CM arrays $\boldsymbol{x}$ and $\boldsymbol{y}$, and scalars α and β, the single-instance inner product routines perform the operations listed below. In these formulas, the inner product occurs over *all* axes of the arrays $\boldsymbol{x}$ and $\boldsymbol{y}$.

Routine	Operation	Data Types
gbl_gen_inner_product	$\alpha = \alpha + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$	real or complex
gbl_gen_inner_product_noadd	$\alpha = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$	real or complex
gbl_gen_inner_product_addto	$\alpha = \beta + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$	real or complex

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

gbl_gen_inner_product_c1	$\alpha = \alpha + \boldsymbol{x}^{H}\boldsymbol{y}$	complex only
gbl_gen_inner_product_c1_noadd	$\alpha = \boldsymbol{x}^{H}\boldsymbol{y}$	complex only
gbl_gen_inner_product_c1_addto	$\alpha = \beta + \boldsymbol{x}^{H}\boldsymbol{y}$	complex only

Details are provided in the man page that follows.

Copyright © 1994 Thinking Machines Corporation

Inner Product

The multiple-instance inner product routines compute one or more instances of an inner product of two vectors. The single-instance (**gbl_**) inner product routines compute the global inner product over all axes of two source CM arrays.

SYNTAX

gen_inner_product	(*z, **x**, **y**, x_vector_axis, y_vector_axis, ier*)
gen_inner_product_noadd	(*z, **x**, **y**, x_vector_axis, y_vector_axis, ier*)
gen_inner_product_addto	(*z, **x**, **y**, **u**, x_vector_axis, y_vector_axis, ier*)
gen_inner_product_c1	(*z, **x**, **y**, x_vector_axis, y_vector_axis, ier*)
gen_inner_product_c1_noadd	(*z, **x**, **y**, x_vector_axis, y_vector_axis, ier*)
gen_inner_product_c1_addto	(*z, **x**, **y**, **u**, x_vector_axis, y_vector_axis, ier*)
gbl_gen_inner_product	(α, *x, y, ier*)
gbl_gen_inner_product_noadd	(α, *x, y, ier*)
gbl_gen_inner_product_addto	(α, *x, y,* β, *ier*)
gbl_gen_inner_product_c1	(α, *x, y, ier*)
gbl_gen_inner_product_c1_noadd	(α, *x, y, ier*)
gbl_gen_inner_product_c1_addto	(α, *x, y,* β, *ier*)

ARGUMENTS

z — CM array of the same data type and precision as ***x*** and ***y***, and rank one less than that of ***x*** and ***y***. The axes of ***z*** must match the instance axes of ***x*** and ***y*** in order of declaration and extents. Thus, each pair of vectors in ***x*** and ***y***, respectively, corresponds to a single value *z* in ***z***.

Copyright © 1994 Thinking Machines Corporation

x	When you call one of the multiple-instance (**gen_**) routines, ***x*** must be a real or complex CM array of rank $\geq$ 2, with at least one non-local instance axis. 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 extent 1.) Axis *x_vector_axis* of ***x*** and axis *y_vector_axis* of ***y*** must have the same extent. The remaining axes of ***x*** and ***y*** (the instance axes) must match in order of declaration and extents. When you call one of the single-instance (**gbl_gen_**) routines, ***x*** must be a real or complex CM array of rank $\geq$ 1.
y	When you call one of the multiple-instance (**gen_**) routines, ***y*** must be a CM array of the same rank and data type as ***x***, with at least one non-local instance axis. 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 extent 1.) Axis *x_vector_axis* of ***x*** and axis *y_vector_axis* of ***y*** must have the same extent. The remaining axes of ***x*** and ***y*** (the instance axes) must match in order of declaration and extents. When you call one of the single-instance (**gbl_gen_**) routines, ***y*** must be a CM array of the same data type, precision, rank, axis extents, and layout as ***x***.
u	CM array of the same data type as ***x*** and ***y***, rank one less than that of ***x*** and ***y***, and the same shape and layout as ***z***. The axes of ***u*** must match the instance axes of ***x*** and ***y*** in order of declaration and extents. Thus, each pair of vectors *x* and *y* in ***x*** and ***y***, respectively, corresponds to a single value *u* in ***u***.
α	Front-end scalar variable of the same data type as ***x*** and ***y***.
β	Front-end scalar variable of the same data type as ***x*** and ***y***.
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.

Copyright © 1994 Thinking Machines Corporation

ier Scalar integer variable. Return code. Upon successful return, or if the CMSSL safety mechanism is turned off, the routines set *ier* to 0.

If the CMSSL safety mechanism is turned on and an error occurs, the multiple-instance (**gen_**) routines set *ier* as follows:

−1 ***z***, ***x***, and ***y*** (and ***u***, in **_addto** calls) are not all valid CM arrays.

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

−3 Axis *x_vector_axis* of ***x*** and axis *y_vector_axis* of ***y*** do not have the same extent.

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

−8 ***z*** and ***u*** do not have the same shape and layout.

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

−11 The data type is not real or complex (single or double precision).

−12 You called **gen_inner_product_c1**, **gen_inner_product_c1_noadd**, or **gen_inner_product_c1_addto**, but supplied data of a type other than complex.

−13 *x_vector_axis* or *y_vector_axis* is a bad axis number (it must be at least 1 and at most equal to the rank of the corresponding array).

−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 order of declaration and extents.

If the CMSSL safety mechanism is turned on and an error occurs, the single-instance (**gbl_gen_**) routines set *ier* as follows:

−1 ***x*** and ***y*** are not both valid CM arrays.

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

−5 ***x*** and ***y*** do not have the same shape and layout.

Copyright © 1994 Thinking Machines Corporation

−30 ***x*** and ***y*** do not have the same data type and precision.

−31 The data type is not real or complex (single or double precision).

−32 You called **gbl_gen_inner_product_c1**, **gbl_gen_inner_product_c1_noadd**, or **gbl_gen_inner_product_c1_addto**, but supplied data of a type other than complex.

DESCRIPTION

Multiple-Instance Routines. The multiple-instance inner product routines perform the operations listed below. The inner product either overwrites the destination CM array, is added to the destination CM array, or is added to a second CM array (with the results placed in the destination CM array).

Routine	Operation	Data Types
gen_inner_product	$z = z + x^{\mathrm{T}}y$	real or complex
gen_inner_product_noadd	$z = x^{\mathrm{T}}y$	real or complex
gen_inner_product_addto	$z = u + x^{\mathrm{T}}y$	real or complex
gen_inner_product_c1	$z = z + x^{\mathrm{H}}y$	complex only
gen_inner_product_c1_noadd	$z = x^{\mathrm{H}}y$	complex only
gen_inner_product_c1_addto	$z = u + x^{\mathrm{H}}y$	complex only

These routines require the source CM arrays to be at least two-dimensional, with at least one non-local instance axis. (The reason for this restriction is that the destination array must have rank one less than that of the source CM arrays, but must also be a CM array — and therefore not completely local.) Thus, to compute the inner product of a *single* pair of vectors, you must either declare any instance axes to have extent 1, or use the single-instance inner product routines.

Upon successful completion of **gen_inner_product** or **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.

Copyright © 1994 Thinking Machines Corporation

Upon successful completion of **gen_inner_product_noadd** or **gen_inner_product_c1_noadd**, the inner product of each vector pair x and y in $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, overwrites the corresponding value in $\boldsymbol{z}$.

Upon successful completion of **gen_inner_product_addto** or **gen_inner_product_c1_addto**, the inner product of each vector pair x and y in $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively (added to the corresponding value in $\boldsymbol{u}$) overwrites the corresponding value in $\boldsymbol{z}$.

Single-Instance Routines. The single-instance inner product routines perform the operations listed below. In these formulas, the inner product occurs over *all* axes of the arrays $\boldsymbol{x}$ and $\boldsymbol{y}$. The inner product either overwrites the destination front-end scalar variable, is added to the destination front-end scalar variable, or is added to a second front-end scalar variable (with the results placed in the destination front-end scalar variable).

Routine	Operation	Data Types
gbl_gen_inner_product	$\alpha = \alpha + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$	real or complex
gbl_gen_inner_product_noadd	$\alpha = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$	real or complex
gbl_gen_inner_product_addto	$\alpha = \beta + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$	real or complex
gbl_gen_inner_product_c1	$\alpha = \alpha + \boldsymbol{x}^{\mathrm{H}}\boldsymbol{y}$	complex only
gbl_gen_inner_product_c1_noadd	$\alpha = \boldsymbol{x}^{\mathrm{H}}\boldsymbol{y}$	complex only
gbl_gen_inner_product_c1_addto	$\alpha = \beta + \boldsymbol{x}^{\mathrm{H}}\boldsymbol{y}$	complex only

Upon successful completion of **gbl_gen_inner_product** or **gbl_gen_inner_product_c1**, the global inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ is added to α.

Upon successful completion of **gbl_gen_inner_product_noadd** or **gbl_gen_inner_product_c1_noadd**, the global inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ overwrites α.

Upon successful completion of **gbl_gen_inner_product_addto** or **gbl_gen_inner_product_c1_addto**, the global inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ (added to β) overwrites α.

Copyright © 1994 Thinking Machines Corporation

NOTES

Overlapping Variables. The arrays ***x*** and ***y*** may be the same variable; the arrays ***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 **gen_inner_product**, **gen_inner_product_c1**, **gen_inner_product_addto**, and **gen_inner_product_c1_addto**
- $(2q-1)I$ for **gen_inner_product_noadd** and **gen_inner_product_c1_noadd**

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

- $8qI$ for **gen_inner_product**, **gen_inner_product_c1**, **gen_inner_product_addto**, and **gen_inner_product_c1_addto**
- $(8q-2)I$ for **gen_inner_product_noadd** and **gen_inner_product_c1_noadd**

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

- $2Q$ for **gbl_gen_inner_product**, **gbl_gen_inner_product_c1**, **gbl_gen_inner_product_addto**, and **gbl_gen_inner_product_c1_addto**
- $2Q-1$ for **gbl_gen_inner_product_noadd** and **gbl_gen_inner_product_c1_noadd**

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

- $8Q$ for **gbl_gen_inner_product**, **gbl_gen_inner_product_c1**, **gbl_gen_inner_product_addto**, and **gbl_gen_inner_product_c1_addto**
- $8Q-2$ for **gbl_gen_inner_product_noadd** and **gbl_gen_inner_product_c1_noadd**

Copyright © 1994 Thinking Machines Corporation

EXAMPLES

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

```
inner-product/cmf/
```

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

Copyright © 1994 Thinking Machines Corporation

3.2 2-Norm

The *multiple-instance* 2-norm routine, **gen_2_norm**, computes one or more instances of the 2-norm of a vector. Given a CM array $\boldsymbol{x}$ containing multiple instances of a vector x, **gen_2_norm** performs the following operation for each instance:

Data Type	Operation
real	$z = (x^{\mathrm{T}}x)^{1/2} = \|\|x\|\|_2$
complex	$z = (x^{\mathrm{H}}x)^{1/2} = \|\|x\|\|_2$

The *single-instance* 2-norm routine, **gbl_gen_2_norm**, computes the global 2-norm of a CM array as defined below. In these formulas, the norm is computed over all axes of the array $\boldsymbol{x}$.

Data Type	Operation
real	$\alpha = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{1/2} = \|\|\boldsymbol{x}\|\|_2$
complex	$\alpha = (\boldsymbol{x}^{\mathrm{H}}\boldsymbol{x})^{1/2} = \|\|\boldsymbol{x}\|\|_2$

The norm is always a real value. Details are provided in the man page that follows.

Copyright © 1994 Thinking Machines Corporation

2-Norm

The multiple-instance 2-norm routine, **gen_2_norm**, computes one or more instances of the 2-norm of a vector. The single-instance 2-norm routine, **gbl_gen_2_norm**, computes the global 2-norm of a CM array.

SYNTAX

gen_2_norm	(*z*, ***x***, *x_vector_axis, ier*)
gbl_gen_2_norm	(α, ***x***, *ier*)

ARGUMENTS

z — Real CM array of the same precision as ***x*** and rank one less than that of ***x***. The axes of *z* must match the instance axes of ***x*** in order of declaration and extents. Thus, each vector *x* in ***x*** corresponds to a single value *z* in *z*.

α — Real front-end scalar variable.

x — When you call **gen_2_norm**, ***x*** must be a real or complex CM array of rank $\geq$ 2, with at least one non-local instance axis. It 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 extent 1.)

When you call **gbl_gen_2_norm**, ***x*** must be a real or complex CM array of rank $\geq$ 1.

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

ier — Scalar integer variable. Return code. Upon successful return, or if the CMSSL safety mechanism is turned off, the routines set *ier* to 0.

If the CMSSL safety mechanism is turned on and an error occurs, **gen_2_norm** sets *ier* as follows:

−1 *z* and ***x*** are not valid CM arrays.

Copyright © 1994 Thinking Machines Corporation

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

−13 *x_vector_axis* is a bad axis number (it must be at least 1 and at most equal to 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 order of declaration and extents.

−40 ***z*** and ***x*** do not have the same precision.

−41 ***z*** has a data type other than real.

−42 ***x*** has a data type other than real or complex.

If the CMSSL safety mechanism is turned on and an error occurs, **gbl_gen_2_norm** sets *ier* as follows:

−1 ***x*** is not a valid CM array.

−31 The data type is not real or complex (single or double precision).

DESCRIPTION

Multiple-Instance Routine. For each instance, **gen_2_norm** performs the following operation:

Data Type	Operation
real	$z = (x^{\mathrm{T}}x)^{1/2} = \|\|x\|\|_2$
complex	$z = (x^{\mathrm{H}}x)^{1/2} = \|\|x\|\|_2$

The **gen_2_norm** routine requires the source CM array to be at least two-dimensional, with at least one non-local instance axis. (The reason for this restriction is that the destination array must have rank one less than that of the source CM array, but must also be a CM array — and therefore not completely local.) Thus, to compute the 2-norm of a *single* vector, you must either declare any instance axes to have extent 1, or use the single-instance 2-norm routine.

Upon successful completion of **gen_2_norm**, the 2-norm of each vector in ***x*** overwrites the corresponding value in ***z***.

Copyright © 1994 Thinking Machines Corporation

Single-Instance Routine. The **gbl_gen_2_norm** routine performs the operations listed below. In these formulas, the norm is computed over all axes of the array $\boldsymbol{x}$.

Data Type	Operation
real	$\alpha = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{1/2} = \|\|\boldsymbol{x}\|\|_2$
complex	$\alpha = (\boldsymbol{x}^{\mathrm{H}}\boldsymbol{x})^{1/2} = \|\|\boldsymbol{x}\|\|_2$

Upon successful completion of **gbl_gen_2_norm**, the global 2-norm of $\boldsymbol{x}$ overwrites α.

NOTES

Numerical Complexity: Multiple-Instance Routine. If the vectors contained in $\boldsymbol{x}$ have length q, then for I instances, the number of floating-point operations used by **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 extents of $\boldsymbol{x}$ is Q, then the number of floating-point operations used by **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 CM Fortran code that uses the 2-norm routines can be found on-line in the subdirectory

```
inner-product/cmf/
```

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

Copyright © 1994 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 CM array, is added to the destination CM array, or is added to a second CM array.

Given CM arrays ***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.

Routine	Operation	Data Types
gen_outer_product	$A = A + xy^{\mathrm{T}}$	real or complex
gen_outer_product_noadd	$A = xy^{\mathrm{T}}$	real or complex
gen_outer_product_addto	$A = B + xy^{\mathrm{T}}$	real or complex
gen_outer_product_c2	$A = A + xy^{\mathrm{H}}$	complex only
gen_outer_product_c2_noadd	$A = xy^{\mathrm{H}}$	complex only
gen_outer_product_c2_addto	$A = B + xy^{\mathrm{H}}$	complex only

The man page following this section provides details.

Copyright © 1994 Thinking Machines Corporation

Outer Product

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

SYNTAX

gen_outer_product	(***A***, ***x***, ***y***, *row_axis, col_axis, x_vector_axis, y_vector_axis, ier*)
gen_outer_product_noadd	(***A***, ***x***, ***y***, *row_axis, col_axis, x_vector_axis, y_vector_axis, ier*)
gen_outer_product_addto	(***A***, ***x***, ***y***, ***B***, *row_axis, col_axis, x_vector_axis, y_vector_axis, ier*)
gen_outer_product_c2	(***A***, ***x***, ***y***, *row_axis, col_axis, x_vector_axis, y_vector_axis, ier*)
gen_outer_product_c2_noadd	(***A***, ***x***, ***y***, *row_axis, col_axis, x_vector_axis, y_vector_axis, ier*)
gen_outer_product_c2_addto	(***A***, ***x***, ***y***, ***B***, *row_axis, col_axis, x_vector_axis, y_vector_axis, ier*)

ARGUMENTS

A CM array of type real or complex and rank greater than or equal to 2. 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 CM array of the same type and precision as ***A*** and rank one less than that of ***A***. 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 extent 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***.

Copyright © 1994 Thinking Machines Corporation

y CM array of the same type and precision as ***A*** and rank one less than that of ***A***. 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 extent 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 CM array of the same type, precision, rank, shape, and layout as ***A***. 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 **gen_outer_product_addto** and **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 between 1 and the rank of ***A***. The axis of ***A*** and ***B*** that counts the rows of the embedded matrix or matrices.

col_axis Scalar integer between 1 and 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 between 1 and the rank of ***x***. The axis of ***x*** along which the elements of each embedded vector lie.

y_vector_axis Scalar integer between 1 and the rank of ***y***. The axis of ***y*** along which the elements of each embedded vector lie.

ier Scalar integer variable. Return code. Upon successful return, or if the CMSSL safety mechanism is turned off, the routines set *ier* to 0.

If the CMSSL safety mechanism is turned on and an error occurs, *ier* is set as follows:

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

Copyright © 1994 Thinking Machines Corporation

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

−4 ***A***, ***x***, ***y***, and ***B*** are not all the same data type (real or complex), or you supplied non-complex data when calling one of the conjugate (**_c2**) routines.

−8 The geometry of ***B*** differs from the geometry of ***A***, or 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 less than 1 or greater than the rank of the associated CM array.

−32 ***A***, ***x***, ***y***, or ***B*** is not a CM array.

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
gen_outer_product	$A = A + xy^{\mathrm{T}}$	real or complex
gen_outer_product_noadd	$A = xy^{\mathrm{T}}$	real or complex
gen_outer_product_addto	$A = B + xy^{\mathrm{T}}$	real or complex
gen_outer_product_c2	$A = A + xy^{\mathrm{H}}$	complex only
gen_outer_product_c2_noadd	$A = xy^{\mathrm{H}}$	complex only
gen_outer_product_c2_addto	$A = B + xy^{\mathrm{H}}$	complex only

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

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

and **gen_outer_product_c2** computes

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

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

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

NOTES

Distinct Variables. ***A*** must be a variable distinct from the source arrays, ***x*** and ***y***. The source arrays can be the same variable, and ***B*** and ***A*** can be 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 extents (p x q), axis *x_vector_axis* of ***x*** has extent p, and axis *y_vector_axis* of ***y*** has extent q, then for I instances, the number of floating-point operations for real operands is

- $2pqI$ for **gen_outer_product** and **gen_outer_product_addto**
- pqI for **gen_outer_product_noadd**

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

- $8pqI$ for **gen_outer_product** and **gen_outer_product_addto**
- $6pqI$ for **gen_outer_product_noadd**

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

EXAMPLES

Sample CM Fortran code that uses the outer product routines can be found on-line in the subdirectory

```
outer-product/cmf/
```

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

Copyright © 1994 Thinking Machines Corporation

3.4 Matrix Vector Multiplication

The matrix vector multiplication routines compute one or more matrix vector products. Given CM arrays ***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.

Routine	Operation	Data Types
gen_matrix_vector_mult	$y = y + Ax$	real or complex
gen_matrix_vector_mult_noadd	$y = Ax$	real or complex
gen_matrix_vector_mult_addto	$y = v + Ax$	real or complex
gen_matrix_vector_mult_c1	$y = y + \overline{A}x$	complex only
gen_matrix_vector_mult_c1_noadd	$y = \overline{A}x$	complex only
gen_matrix_vector_mult_c1_addto	$y = v + \overline{A}x$	complex only

The man page following this section provides details.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

Matrix Vector Multiplication

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

SYNTAX

gen_matrix_vector_mult	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_matrix_vector_mult_noadd	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_matrix_vector_mult_addto	(***y***, ***A***, ***x***, ***v***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_matrix_vector_mult_c1	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_matrix_vector_mult_c1_noadd	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_matrix_vector_mult_c1_addto	(***y***, ***A***, ***x***, ***v***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)

ARGUMENTS

y CM array of rank greater than or equal to 1 and type real or complex. 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 extent as axis *row_axis* of ***A***. Upon completion, each vector instance is overwritten by the result of the matrix vector multplication call.

A CM array of the same type and precision as ***y*** and rank one greater than that of ***y***. 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.

Copyright © 1994 Thinking Machines Corporation

x CM array of the same rank, type, and precision as ***y***. 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 extent 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 CM array of the same rank, type, precision, shape, and layout as ***y***. This argument is used only in the **gen_matrix_vector_mult_addto** and **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 between 1 and the rank of ***y***. The axis of ***y*** and ***v*** along which the elements of the embedded vectors lie.

row_axis Scalar integer between 1 and the rank of ***A***. The axis of ***A*** that counts the rows of the embedded matrix or matrices. (For definitions of row and column axes, see Chapter 1.)

col_axis Scalar integer between 1 and the rank of ***A***. The axis of ***A*** that counts the columns of the embedded matrix or matrices. (For definitions of row and column axes, see Chapter 1.)

x_vector_axis Scalar integer between 1 and the rank of ***x***. The axis of ***x*** along which the elements of the embedded vectors lie.

ier Scalar integer variable. Return code. Upon successful return, or if the CMSSL safety mechanism is turned off, the routines set *ier* to 0.

If the CMSSL safety mechanism is turned on and an error occurs, *ier* is set 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 extent, or axis *col_axis* of ***A*** and axis *x_vector_axis* of ***x*** do not have the same extent.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

−4 Matrix or vectors are not of the same data type (real or complex); or you supplied non-complex data when calling one of the conjugate (**_c1**) routines.

−8 Instance axes of the input CM arrays do not match in length and order of declaration; or ***y*** and ***v*** do not have the same rank, shape, and layout.

−32 ***A***, ***x***, ***y***, or ***v*** is not a CM array.

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
gen_matrix_vector_mult	$y = y + Ax$	real or complex
gen_matrix_vector_mult_noadd	$y = Ax$	real or complex
gen_matrix_vector_mult_addto	$y = v + Ax$	real or complex
gen_matrix_vector_mult_c1	$y = y + \overline{A}x$	complex only
gen_matrix_vector_mult_c1_noadd	$y = \overline{A}x$	complex only
gen_matrix_vector_mult_c1_addto	$y = v + \overline{A}x$	complex only

NOTES

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

Numerical Stability. The algorithm is numerically stable.

Copyright © 1994 Thinking Machines Corporation

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

	Real Operands	Complex Operands
gen_matrix_vector_mult	$2pqI$	$8pqI$
gen_matrix_vector_mult_addto	$2pqI$	$8pqI$
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 CM Fortran code that uses the matrix vector multiplication routines can be found on-line in the subdirectory

```
matrix-vector/cmf/
```

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

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

3.5 Vector Matrix Multiplication

The vector matrix multiplication routines compute one or more vector matrix products. Given CM arrays ***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.

Routine	Operation	Data Types
gen_vector_matrix_mult	$y^{\mathrm{T}} = y^{\mathrm{T}} + x^{\mathrm{T}}A$	real or complex
gen_vector_matrix_mult_noadd	$y^{\mathrm{T}} = x^{\mathrm{T}}A$	real or complex
gen_vector_matrix_mult_addto	$y^{\mathrm{T}} = v^{\mathrm{T}} + x^{\mathrm{T}}A$	real or complex
gen_vector_matrix_mult_c2	$y^{\mathrm{T}} = y^{\mathrm{T}} + x^{\mathrm{T}}\overline{A}$	complex only
gen_vector_matrix_mult_c2_noadd	$y^{\mathrm{T}} = x^{\mathrm{T}}\overline{A}$	complex only
gen_vector_matrix_mult_c2_addto	$y^{\mathrm{T}} = v^{\mathrm{T}} + x^{\mathrm{T}}\overline{A}$	complex only

The man page following this section provides details.

Copyright © 1994 Thinking Machines Corporation

Vector Matrix Multiplication

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

SYNTAX

gen_vector_matrix_mult	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_vector_matrix_mult_noadd	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_vector_matrix_mult_addto	(***y***, ***A***, ***x***, ***v***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_vector_matrix_mult_c2	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_vector_matrix_mult_c2_noadd	(***y***, ***A***, ***x***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)
gen_vector_matrix_mult_c2_addto	(***y***, ***A***, ***x***, ***v***, *y_vector_axis, row_axis, col_axis, x_vector_axis, ier*)

ARGUMENTS

y CM array of rank greater than or equal to 1 and type real or complex. 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 extent as axis *col_axis* of ***A***. Upon successful completion, each vector instance is overwritten by the result of the vector matrix multplication call.

A CM array of the same type and precision as ***y*** and rank one greater than that of ***y***. 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.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

x CM array of the same rank, type, and precision as ***y***. 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 extent 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 CM array of the same rank, type, precision, shape, and layout as ***y***. This argument is used only in the **gen_vector_matrix_mult_addto** and **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 between 1 and the rank of ***y***. The axis of ***y*** and ***v*** along which the elements of the embedded vectors lie.

row_axis Scalar integer between 1 and the rank of ***A***. The axis of ***A*** that counts the rows of the embedded matrix or matrices. (For definitions of row and column axes, see Chapter 1.)

col_axis Scalar integer between 1 and the rank of ***A***. The axis of ***A*** that counts the columns of the embedded matrix or matrices. (For definitions of row and column axes, see Chapter 1.)

x_vector_axis Scalar integer between 1 and the rank of ***x***. The axis of ***x*** along which the elements of each vector lie.

ier Scalar integer variable. Return code. Upon successful return, or if the CMSSL safety mechanism is turned off, the routines set *ier* to 0.

If the CMSSL safety mechanism is turned on and an error occurs, *ier* is set 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 extent, or axis *row_axis* of ***A*** and axis *x_vector_axis* of ***x*** do not have the same extent.

Copyright © 1994 Thinking Machines Corporation

−4 Matrix or vectors are not of the same data type (real or complex); or you supplied non-complex data when calling one of the conjugate (**_c2**) routines.

−8 Instance axes of the input CM arrays do not match in length and order of declaration; or ***y*** and ***v*** do not have the same rank, shape, and layout.

−32 ***A***, ***x***, ***y***, or ***v*** is not a CM array.

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
gen_vector_matrix_mult	$y^T = y^T + x^T A$	real or complex
gen_vector_matrix_mult_noadd	$y^T = x^T A$	real or complex
gen_vector_matrix_mult_addto	$y^T = v^T + x^T A$	real or complex
gen_vector_matrix_mult_c2	$y^T = y^T + x^T \overline{A}$	complex only
gen_vector_matrix_mult_c2_noadd	$y^T = x^T \overline{A}$	complex only
gen_vector_matrix_mult_c2_addto	$y^T = v^T + x^T \overline{A}$	complex only

NOTES

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

Numerical Stability. The algorithm is numerically stable.

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

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

	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 CM Fortran code that uses the vector matrix multiplication routines can be found on-line in the subdirectory

```
vector-matrix/cmf/
```

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

Copyright © 1994 Thinking Machines Corporation

3.6 Infinity Norm

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

The infinity norm of a matrix A^{-1} can be estimated based on the QR or LU factors of A using the method developed by Hager (see reference 10 listed in Section 3.9). The **gen_lu_infinity_norm_inv** and **gen_qr_infinity_norm_inv** routines, described in Chapter 5, perform these estimations.

Copyright © 1994 Thinking Machines Corporation

Infinity Norm

Given one or more matrices embedded in a CM array, the **gen_infinity_norm** routine computes the infinity norms of the matrices.

SYNTAX

gen_infinity_norm (***A_norm***, *A, m, n, row_axis, col_axis, ier*)

ARGUMENTS

A_norm Real CM array with the same rank and precision as ***A***. Axes *row_axis* and *col_axis* must have extent 1.

Upon successful completion of **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 x 16 x 4 x 128, with *row_axis* = 2 and *col_axis* = 3, then upon completion, ***A_norm***(*r*, 1, 1, *s*) contains the infinity norm of the matrix consisting of ***A***(*r*, :, :, *s*).

A Real or complex CM array of rank $\geq$ 2.

When you call **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 extents 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***.

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. The axis that counts the rows of the matrices *A* embedded in ***A***. (For definitions of row and column axes, see Chapter 1.)

Copyright © 1994 Thinking Machines Corporation

col_axis Scalar integer variable. The axis that counts the columns of the matrices A embedded in ***A***. (For definitions of row and column axes, see Chapter 1.)

ier Scalar integer variable. Return code; set to 0 upon successful return, or to 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***.
−32	***A*** is not real or complex, ***A_norm*** is not real, or ***A*** and ***A_norm*** do not have the same precision.
−64	*row_axis* or *col_axis* is invalid.

DESCRIPTION

Given one or more matrices A embedded in a CM array ***A***, the **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:

$$||x||_\infty = \max_i |x_i|$$

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

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

EXAMPLES

Sample CM Fortran code that uses the **gen_infinity_norm** routine can be found on-line in the subdirectory

```
infinity-norm/cmf/
```

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

Copyright © 1994 Thinking Machines Corporation

3.7 Matrix Multiplication

The matrix multiplication routines compute one or more matrix products. Given CM arrays ***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 algorithm used depends on the axis extents of the arrays supplied. The man page following this section provides details about this routine.

Routine	Operation	Data Types
gen_matrix_mult	$C = C + AB$	real or complex
gen_matrix_mult_noadd	$C = AB$	real or complex
gen_matrix_mult_addto	$C = D + AB$	real or complex
gen_matrix_mult_t1	$C = C + A^{\mathrm{T}}B$	real or complex
gen_matrix_mult_t1_noadd	$C = A^{\mathrm{T}}B$	real or complex
gen_matrix_mult_t1_addto	$C = D + A^{\mathrm{T}}B$	real or complex
gen_matrix_mult_h1	$C = C + A^{\mathrm{H}}B$	complex only
gen_matrix_mult_h1_noadd	$C = A^{\mathrm{H}}B$	complex only
gen_matrix_mult_h1_addto	$C = D + A^{\mathrm{H}}B$	complex only
gen_matrix_mult_t2	$C = C + AB^{\mathrm{T}}$	real or complex
gen_matrix_mult_t2_noadd	$C = AB^{\mathrm{T}}$	real or complex
gen_matrix_mult_t2_addto	$C = D + AB^{\mathrm{T}}$	real or complex
gen_matrix_mult_h2	$C = C + AB^{\mathrm{H}}$	complex only
gen_matrix_mult_h2_noadd	$C = AB^{\mathrm{H}}$	complex only
gen_matrix_mult_h2_addto	$C = D + AB^{\mathrm{H}}$	complex only
gen_matrix_mult_t1_t2	$C = C + A^{\mathrm{T}}B^{\mathrm{T}}$	real or complex
gen_matrix_mult_t1_t2_noadd	$C = A^{\mathrm{T}}B^{\mathrm{T}}$	real or complex
gen_matrix_mult_t1_t2_addto	$C = D + A^{\mathrm{T}}B^{\mathrm{T}}$	real or complex

Copyright © 1994 Thinking Machines Corporation

Matrix Multiplication

The matrix multiplication routines compute one or more matrix products.

SYNTAX

gen_matrix_mult	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_noadd	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_addto	(***C***, ***A***, ***B***, ***D***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t1	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t1_noadd	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t1_addto	(***C***, ***A***, ***B***, ***D***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_h1	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_h1_noadd	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_h1_addto	(***C***, ***A***, ***B***, ***D***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t2	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t2_noadd	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t2_addto	(***C***, ***A***, ***B***, ***D***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_h2	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_h2_noadd	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_h2_addto	(***C***, ***A***, ***B***, ***D***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t1_t2	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t1_t2_noadd	(***C***, ***A***, ***B***, *row_axis*, *col_axis*, *ier*)
gen_matrix_mult_t1_t2_addto	(***C***, ***A***, ***B***, ***D***, *row_axis*, *col_axis*, *ier*)

Copyright © 1994 Thinking Machines Corporation

ARGUMENTS

C CM array of type real or complex and rank greater than or equal to 2. 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 extent as axis *row_axis* of ***A***. Axis *col_axis* of ***C*** must have the same extent 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 CM array of the same rank, type, and precision as ***C***. 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 extent as axis *row_axis* of ***B***. The contents of ***A*** are not changed during execution.

B CM array of the same rank, type, and precision as ***C***. 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 CM array of the same rank, type, precision, shape, and layout 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 between 1 and the rank of ***C***. The axis of ***C***, ***A***, ***B***, and ***D*** that counts the rows of the embedded matrix or matrices. (For definitions of row and column axes, see Chapter 1.)

col_axis Scalar integer between 2 and the rank of ***C***. The axis of ***C***, ***A***, ***B***, and ***D*** that counts the columns of the embedded matrix or matrices. (For definitions of row and column axes, see Chapter 1.)

ier Scalar integer variable. Return code. Upon successful return, or if the CMSSL safety mechanism is turned off, the routines set *ier* to 0.

Copyright © 1994 Thinking Machines Corporation

If the CMSSL safety mechanism is turned on and an error occurs, *ier* is set as follows:

−1	Ranks of provided arrays are different or are not at least 2.
−2	Non-conforming ***A*** and ***B*** axis extents.
−4	Non-conforming ***A*** and ***C*** axis extents.
−8	Non-conforming ***B*** and ***C*** axis extents.
−16	The instance axes of ***A***, ***B***, and ***C*** do not match in length and order of declaration; or ***C*** and ***D*** do not have the same rank, shape, and layout.
−32	*row_axis* and/or *col_axis* is less than 1 or greater than he rank of the arrays.
−64	***C***, ***A***, ***B***, or ***D*** is not a CM array.
−128	***C***, ***A***, and ***B*** (and ***D***, in **_addto** calls) do not all have the same data type (real or complex), or you supplied non-complex data when calling one of the Hermitian (**_h1** or **_h2**) routines.

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
gen_matrix_mult	$C = C + AB$	real or complex
gen_matrix_mult_noadd	$C = AB$	real or complex
gen_matrix_mult_addto	$C = D + AB$	real or complex
gen_matrix_mult_t1	$C = C + A^{\mathrm{T}}B$	real or complex
gen_matrix_mult_t1_noadd	$C = A^{\mathrm{T}}B$	real or complex
gen_matrix_mult_t1_addto	$C = D + A^{\mathrm{T}}B$	real or complex

Copyright © 1994 Thinking Machines Corporation

gen_matrix_mult_h1	$C = C + A^{\mathrm{H}}B$	complex only
gen_matrix_mult_h1_noadd	$C = A^{\mathrm{H}}B$	complex only
gen_matrix_mult_h1_addto	$C = D + A^{\mathrm{H}}B$	complex only
gen_matrix_mult_t2	$C = C + AB^{\mathrm{T}}$	real or complex
gen_matrix_mult_t2_noadd	$C = AB^{\mathrm{T}}$	real or complex
gen_matrix_mult_t2_addto	$C = D + AB^{\mathrm{T}}$	real or complex
gen_matrix_mult_h2	$C = C + AB^{\mathrm{H}}$	complex only
gen_matrix_mult_h2_noadd	$C = AB^{\mathrm{H}}$	complex only
gen_matrix_mult_h2_addto	$C = D + AB^{\mathrm{H}}$	complex only
gen_matrix_mult_t1_t2	$C = C + A^{\mathrm{T}}B^{\mathrm{T}}$	real or complex
gen_matrix_mult_t1_t2_noadd	$C = A^{\mathrm{T}}B^{\mathrm{T}}$	real or complex
gen_matrix_mult_t1_t2_addto	$C = D + A^{\mathrm{T}}B^{\mathrm{T}}$	real or complex

The algorithm used depends on the axis extents of the arrays supplied.

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

Array	row_axis extent	col_axis extent
A	p	q
B	q	r
C	p	r
D	p	r

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

Copyright © 1994 Thinking Machines Corporation

Array	row_axis extent	col_axis extent
A	*q*	*p*
B	*q*	*r*
C	*p*	*r*
D	*p*	*r*

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

Array	row_axis extent	col_axis extent
A	*p*	*q*
B	*r*	*q*
C	*p*	*r*
D	*p*	*r*

For calls that transpose both *A* and *B*, the arrays conform correctly with the following axis extents for *row_axis* and *col_axis*:

Array	row_axis extent	col_axis extent
A	*q*	*p*
B	*r*	*q*
C	*p*	*r*
D	*p*	*r*

NOTES

Distinct Variables. All input arrays must be distinct, except that ***C*** and ***D*** can be the same variable.

Numerical Stability. The algorithm is numerically stable.

Copyright © 1994 Thinking Machines Corporation

Numerical Complexity. If the matrices embedded in A have the axis extents 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 CM Fortran code that uses the matrix multiplication routines can be found on-line in the subdirectory

```
matrix-multiply/cmf/
```

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

Copyright © 1994 Thinking Machines Corporation

3.8 Matrix Multiplication with External Storage

The **gen_matrix_mult_ext** routine performs the operation

$$\boldsymbol{Y} = \boldsymbol{Y} + \boldsymbol{AX}$$

where $\boldsymbol{Y}$ is a matrix of size $n1 \times m$, $\boldsymbol{X}$ is a matrix of size $n2 \times m$, and $\boldsymbol{A}$ is a matrix of size $n1 \times n2$ that is too large to fit into core memory. The man page that follows provides details.

NOTE

This routine does not work with the nodal CMSSL library.

Copyright © 1994 Thinking Machines Corporation

Matrix Multiplication with External Storage

The **gen_matrix_mult_ext** routine performs the operation $Y = Y + AX$ where Y is a matrix of size $n1 \times m$, X is a matrix of size $n2 \times m$, and A is a matrix of size $n1 \times n2$ that is too large to fit into core memory.

SYNTAX

gen_matrix_mult_ext (*Y, X, m, n1, n2, blk, type, unit, ier*)

ARGUMENTS

Y	CM array of rank 2, the same data type as *A* (real or complex), and size *n1* x *m*. Upon return, contains *Y* + *AX*.
X	CM array of rank 2, the same data type as *A*, and size *n2* x *m*.
m	Scalar integer variable. The number of columns in *X* and *Y*.
n1	Scalar integer variable. The number of rows in *A* and *Y*.
n2	Scalar integer variable. The number of rows in *X* and columns in *A*.
blk	Scalar integer variable. Block size. The matrix *A* is partitioned into blocks of *blk* columns, or panels. See the Notes section, below, for guidelines for choosing *blk*.
type	Scalar integer variable. The data type. Specify one of the following values:

CMSSL_single_real	real*4
CMSSL_double_real	real*8
CMSSL_single_complex	complex*8
CMSSL_double_complex	complex*16

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

unit Scalar integer. Valid unit number associated with the file that contains the matrix ***A*** stored in serial order (see the Notes below.) Use the CM Fortran utility **CMF_FILE_OPEN** to associate a file with a unit number (or use the equivalent utility to associate a socket or device with a unit number). The **gen_matrix_mult_ext** routine reads the matrix from *unit* and does not modify the file.

ier Scalar integer variable. Return code. Set to 0 upon successful return, or to −1 if the routine encounters an I/O error on *unit*.

DESCRIPTION

The **gen_matrix_mult_ext** routine performs the operation $\boldsymbol{Y} = \boldsymbol{Y} + \boldsymbol{AX}$ where ***Y*** is a matrix of size *n1* x *m*, ***X*** is a matrix of size *n2* x *m*, and ***A*** is a matrix of size *n1* x *n2* that is too large to fit into core memory.

NOTES

No Nodal Library Support. The **gen_matrix_mult_ext** routine does not work with the nodal CMSSL library.

Include the CMSSL Header File. Because the routine described above uses symbolic constants, you must include the line

```
INCLUDE '/usr/include/cm/cmssl-cmf.h'
```

at the top of any program module that calls these routines. This file declares the types of the CMSSL symbolic constants.

File Unit. The I/O unit *unit* must be assigned to a file before you call **gen_matrix_mult_ext**. In CM Fortran, file assignment is done with the **CMF_FILE_OPEN** utility (or an equivalent utility for a device or socket). For information regarding parallel I/O in general, see the *CM-5 I/O System Programming Guide*. For information about the CM Fortran interface to parallel I/O, see the *CM Fortran Utility Library Reference Manual*. As described in this manual, there are essentially two modes of external storage: Fixed Machine Size (FMS) and Serial Order (SO). Serial order is the familiar Fortran row-major order and is the one used by the external matrix multiplication routine. Therefore, ***A*** must be stored in serial order in file unit *unit*.

Copyright © 1994 Thinking Machines Corporation

In this order, the data is portable across the CM-5 external storage systems (DataVault, Scalable Disk Array, HIPPI).

Choosing the Block Size. The **gen_matrix_mult_ext** routine partitions the matrix A into block columns, or panels, A_i, of size n x blk:

$$A = [\, A_1, A_2, ..., A_m \,].$$

The last panel, A_m, contains fewer than blk columns if blk is not a divisor of n. The block size should be large enough to optimize machine utilization. Besides the allocation of X and Y, the in-core memory requirement for **gen_matrix_mult_ext** is approximately $(2v + 16)n*blk$ bytes, where v is the number of bytes in the data type of A.

EXAMPLES

Sample CM Fortran code that uses the external matrix multiplication routine can be found on-line in the subdirectory

```
external/matrix-multiply/cmf/
```

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

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

3.9 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 © 1994 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 © 1994 Thinking Machines Corporation