C/FORTRAN System Calls Reference

Contents

This section describes the iCC system calls. It is divided into two subsections:

- NX-like whole partition routines
- MPI-like group routines

Included are the calling sequences for both the C and FORTRAN versions, explanations of all the parameters, as well as discussions of the semantics of the operations.

Notational Conventions

Bold	Identifies system call names.
iCC_operation()	Identifies a C system call.
iCC_OPERATION()	Identifies a FORTRAN system call.
Italic	Identifies system call parameters.

Whole Partition Routines

Contents

This section describes the whole partition routines that comprised the original, NX-like iCC library.

iCC_barrier()
iCC_BARRIER()

Loose (faster) synchronization of all nodes.

Synopsis

void **iCC_barrier**()

SUBROUTINE **iCC_BARRIER()**

Discussion

This routine guarantees that all nodes have executed it before any node continues. It does not guarantee that all nodes will leave the routine at the same time. (Use the more expensive **iCC_gsync()/iCC_GSYNC()** routine to insure that all nodes exit the routine simultaneously.)

This is a global operation. All nodes must execute this routine for it to complete.

iCC_bcast()
iCC_BCAST()

Global broadcast operation.

Synopsis

void **iCC_bcast(**
    char *x*[],
    long *xlen*,
    long *iroot***)**

SUBROUTINE **iCC_BCAST(***x, xlen, iroot***)**
    INTEGER *x*(*)
    INTEGER *xlen*
    INTEGER *iroot*

Parameters

x	Pointer to the buffer containing the message to be broadcast/received.
xlen	Size (in bytes) of the message to be broadcast.
iroot	ID of the node whose message is to be broadcast.

Discussion

At the end of the operation, all nodes' buffers contain the message in the root node's buffer.

This operation is functionally equivalent to Intel's **csend()/CSEND()** with the destination node ID set to -1, except that a message type for the broadcast cannot be specified.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dg*x*high()
iCC_DG*x*HIGH()

Global vector MAXIMUM operation with result distributed over the nodes.

Synopsis

void **iCC_dgdhigh(**
    double *x*[],
    double *y*[],
    long *ylens*[])

void **iCC_dgihigh(**
    long *x*[],
    long *y*[],
    long *ylens*[])

void **iCC_dgshigh(**
    float *x*[],
    float *y*[],
    long *ylens*[])

SUBROUTINE **iCC_DGDHIGH(***x, y, ylens***)**
    DOUBLE PRECISION *x*(*)
    DOUBLE PRECISION *y*(*)
    INTEGER *ylens*(*)

SUBROUTINE **iCC_DGIHIGH(***x, y, ylens***)**
    INTEGER *x*(*)
    INTEGER *y*(*)
    INTEGER *ylens*(*)

SUBROUTINE **iCC_DGSHIGH(***x, y, ylens***)**
    SINGLE PRECISION *x*(*)
    SINGLE PRECISION *y*(*)
    INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers, single-precision floats, or double-precision floats)

of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the maximum of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dg*x*low()
iCC_DG*x*LOW()

Global vector MINIMUM operation with result distributed over the nodes.

Synopsis

void **iCC_dgdlow(**
double *x*[],
double *y*[],
long *ylens*[])

void **iCC_dgilow(**
long *x*[],
long *y*[],
long *ylens*[])

void **iCC_dgslow(**
float *x*[],
float *y*[],
long *ylens*[])

SUBROUTINE **iCC_DGDLOW(***x, y, ylens***)**
DOUBLE PRECISION *x*(*)
DOUBLE PRECISION *y*(*)
INTEGER *ylens*(*)

SUBROUTINE **iCC_DGILOW(***x, y, ylens***)**
INTEGER *x*(*)
INTEGER *y*(*)
INTEGER *ylens*(*)

SUBROUTINE **iCC_DGSLOW(***x, y, ylens***)**
SINGLE PRECISION *x*(*)
SINGLE PRECISION *y*(*)
INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers, single-precision floats, or double-precision floats)

of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the minimum of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dg*x*prod()
iCC_DG*x*PROD()

Global vector PRODUCT operation with result distributed over the nodes.

Synopsis

void **iCC_dgdprod(**
    double *x*[],
    double *y*[],
    long *ylens*[])

void **iCC_dgiprod(**
    long *x*[],
    long *y*[],
    long *ylens*[])

void **iCC_dgsprod(**
    float *x*[],
    float *y*[],
    long *ylens*[])

SUBROUTINE **iCC_DGDPROD(***x, y, ylens***)**
    DOUBLE PRECISION *x*(*)
    DOUBLE PRECISION *y*(*)
    INTEGER *ylens*(*)

SUBROUTINE **iCC_DGIPROD(***x, y, ylens***)**
    INTEGER *x*(*)
    INTEGER *y*(*)
    INTEGER *ylens*(*)

SUBROUTINE **iCC_DGSPROD(***x, y, ylens***)**
    SINGLE PRECISION *x*(*)
    SINGLE PRECISION *y*(*)
    INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers, single-precision floats, or double-

precision floats) of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the product of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dg*x*sum()
iCC_DG*x*SUM()

Global vector SUM operation with result distributed over the nodes.

Synopsis

void **iCC_dgdsum(**
double *x*[],
double *y*[],
long *ylens*[])

void **iCC_dgisum(**
long *x*[]
long *y*[],
long *ylens*[])

void **iCC_dgssum(**
float *x*[],
float *y*[],
long *ylens*[])

SUBROUTINE **iCC_DGDSUM(***x, y, ylens***)**
DOUBLE PRECISION *x*(*)
DOUBLE PRECISION *y*(*)
INTEGER *ylens*(*)

SUBROUTINE **iCC_DGISUM(***x, y, ylens***)**
INTEGER *x*(*)
INTEGER *y*(*)
INTEGER *ylens*(*)

SUBROUTINE **iCC_DGSSUM(***x, y, ylens***)**
SINGLE PRECISION *x*(*)
SINGLE PRECISION *y*(*)
INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers, single-precision floats, or double-

precision floats) of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the sum of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dgiand()
iCC_DGIAND()

Global vector bitwise AND operation with result distributed over the nodes.

Synopsis

void **iCC_dgiand(**
long *x*[],
long *y*[],
long *ylens*[])

SUBROUTINE **iCC_DGIAND(***x, y, ylens***)**
INTEGER *x*(*)
INTEGER *y*(*)
INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers) of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the bitwise ANDing of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dgior()
iCC_DGIOR()

Global vector bitwise OR operation with result distributed over the nodes.

Synopsis

void **iCC_dgior(**
long *x*[],
long *y*[],
long *ylens*[])

SUBROUTINE **iCC_DGIOR(***x, y, ylens***)**
INTEGER *x*(*)
INTEGER *y*(*)
INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers) of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the bitwise ORing of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dgland()
iCC_DGLAND()

Global vector logical AND operation with result distributed over the nodes.

Synopsis

void **iCC_dgland(**
long *x*[],
long *y*[],
long *ylens*[])

SUBROUTINE **iCC_DGLAND(***x, y, ylens***)**
LOGICAL *x*(*)
LOGICAL *y*(*)
INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers/LOGICALs) of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the logical ANDing of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dglor()
iCC_DGLOR()

Global vector logical OR operation with result distributed over the nodes.

Synopsis

void **iCC_dglor(**
  long *x*[],
  long *y*[],
  long *ylens*[])

SUBROUTINE **iCC_DGLOR(***x, y, ylens***)**
  LOGICAL *x*(*)
  LOGICAL *y*(*)
  INTEGER *ylens*(*)

Parameters

x	Input vector to be used in the operation.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in integers/LOGICALs) of each node's portion of the result (indexed by node ID).

Discussion

At the end of the operation, each node's *y* contains a portion of the logical ORing of every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_dgopf()
iCC_DGOPF()

User-defined global vector operation with result distributed over the nodes.

Synopsis

```
void iCC_dgopf(
        char x[],
        long size,
        char y[],
        long ylens[],
        long (*f)())
```

```
SUBROUTINE iCC_DGOPF(x, size, y, ylens, f)
        INTEGER x(*)
        INTEGER size
        INTEGER y(*)
        INTEGER ylens(*)
        EXTERNAL f
```

Parameters

x	Input vector to be used in the operation.
size	Size, in bytes, of a unit of the user's data type.
y	Output vector as determined by *ylens*.
ylens	Array containing the length (in units of the user's data type) of each node's portion of the result (indexed by node ID).
f	User-supplied vector function of the following format:

```
void f(
    <any type> w[],
    <any type> v[],
    long *n)
```

```
SUBROUTINE f(w, v, n)
    <any type> w(*)
    <any type> v(*)
    INTEGER n
```

where *w* is an input-only array, *v* is an input-output array (i.e., it is overwritten), and *n* is the number of elements in *w* and *v*. The value of *n* must not be changed by the user function. It is passed by reference only to provide FORTRAN compatibility.

Discussion

At the end of the operation, each node's *y* contains a portion of the application of *f*() to every node's *x*. The length of each node's portion is determined by *ylens*.

This operation has no functional equivalent in Intel's library. Note that the interface to the user-supplied function is different than that for Intel's **gopf()/GOPF()** call.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gcol()
iCC_GCOL()

Global collect (concatenation) operation for contributions of unknown length.

Synopsis

void **iCC_gcol(**
    char *x*[],
    long *xlen*,
    char *y*[],
    long *ylen*,
    long **ncnt***)**

SUBROUTINE **iCC_GCOL(***x, xlen, y, ylen, ncnt***)**
    INTEGER *x*(*)
    INTEGER *xlen*
    INTEGER *y*(*)
    INTEGER *ylen*
    INTEGER *ncnt*

Parameters

x	Pointer to the buffer containing the node's contribution.
xlen	Size (in bytes) of the node's contribution.
y	Pointer to the buffer containing the collected result.
ylen	Size (in bytes) of the node's *y*.
ncnt	Number of bytes actually collected into *y*.

Discussion

At the end of the operation, all nodes' buffers contain the concatenation of their contributions. This routine is slightly less efficient than **iCC_gcolx()/iCC_GCOLX()**.

This operation is functionally equivalent to Intel's **gcol()/GCOL()**.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gcolx()
iCC_GCOLX()

Global collect (concatenation) operation for contributions of known length.

Synopsis

void **iCC_gcolx(**
char *x*[],
long *xlens*[],
char *y*[])

SUBROUTINE **iCC_GCOLX(***x, xlens, y***)**
INTEGER *x*(*)
INTEGER *xlens*(*)
INTEGER *y*(*)

Parameters

x	Pointer to the buffer containing the node's contribution.
xlens	Array containing size (in bytes) of each node's contribution (indexed by node ID).
y	Pointer to the buffer to contain the collected result. (Assumed to be large enough since size of all contributions are known beforehand.)

Discussion

At the end of the operation, all nodes' buffers contain the concatenation of their contributions. This routine is slightly more efficient than **iCC_gcol()/iCC_GCOL()**.

This operation is functionally equivalent to Intel's **gcolx()/GCOLX()**.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*high()
iCC_G*x*HIGH()

Global vector MAXIMUM operation with result on all nodes.

Synopsis

void **iCC_gdhigh(**
    double *x*[],
    long *xlen*,
    double *y*[])

void **iCC_gihigh(**
    long *x*[],
    long *xlen*,
    long *y*[])

void **iCC_gshigh(**
    float *x*[],
    long *xlen*,
    float *y*[])

SUBROUTINE **iCC_GDHIGH(***x, xlen, y***)**
    DOUBLE PRECISION *x*(*)
    INTEGER *xlen*
    DOUBLE PRECISION *y*(*)

SUBROUTINE **iCC_GIHIGH(***x, xlen, y***)**
    INTEGER *x*(*)
    INTEGER *xlen*
    INTEGER *y*(*)

SUBROUTINE **iCC_GSHIGH(***x, xlen, y***)**
    SINGLE PRECISION *x*(*)
    INTEGER *xlen*
    SINGLE PRECISION *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the maximum of every node's *x*.

This operation is functionally equivalent to Intel's **g*x*high()/G*X*HIGH()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*highto1()
iCC_G*x*HIGHTO1()

Global vector MAXIMUM operation with result on one node.

Synopsis

void **iCC_gdhighto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gihighto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gshighto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

SUBROUTINE **iCC_GDHIGHTO1(***x, xlen, y, iroot***)**
DOUBLE PRECISION *x*(*)
INTEGER *xlen*
DOUBLE PRECISION *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GIHIGHTO1(***x, xlen, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GSHIGHTO1(***x, xlen, y, iroot***)**
SINGLE PRECISION *x*(*)
INTEGER *xlen*
SINGLE PRECISION *y*(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers, single-precision floats, or double-precision floats) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the maximum of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*low()
iCC_G*x*LOW()

Global vector MININUM operation with result on all nodes.

Synopsis

void **iCC_gdlow(**
double *x*[],
long *xlen*,
double *y*[])

void **iCC_gilow(**
long *x*[],
long *xlen*,
long *y*[])

void **iCC_gslow(**
float *x*[],
long *xlen*,
float *y*[])

SUBROUTINE **iCC_GDLOW(***x, xlen, y***)**
DOUBLE PRECISION *x*(*)
INTEGER *xlen*
DOUBLE PRECISION *y*(*)

SUBROUTINE **iCC_GILOW(***x, xlen, y***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)

SUBROUTINE **iCC_GSLOW(***x, xlen, y***)**
SINGLE PRECISION *x*(*)
INTEGER *xlen*
SINGLE PRECISION *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the minimum of every node's *x*.

This operation is functionally equivalent to Intel's **g*x*low()/G*X*LOW()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*lowto1()
iCC_G*x*LOWTO1()

Global vector MINIMUM operation with result on one node.

Synopsis

void **iCC_gdlowto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gilowto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gslowto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

SUBROUTINE **iCC_GDLOWTO1(***x, xlen, y, iroot***)**
DOUBLE PRECISION *x*(*)
INTEGER *xlen*
DOUBLE PRECISION *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GILOWTO1(***x, xlen, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GSLOWTO1(***x, xlen, y, iroot***)**
SINGLE PRECISION *x*(*)
INTEGER *xlen*
SINGLE PRECISION *y*(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers, single-precision floats, or double-precision floats) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the minimum of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*prod()
iCC_G*x*PROD()

Global vector PRODUCT operation with result on all nodes.

Synopsis

void **iCC_gdprod(**
double *x*[],
long *xlen*,
double *y*[])

void **iCC_giprod(**
long *x*[],
long *xlen*,
long *y*[])

void **iCC_gsprod(**
float *x*[],
long *xlen*,
float *y*[])

SUBROUTINE **iCC_GDPROD(***x, xlen, y***)**
DOUBLE PRECISION *x*(*)
INTEGER *xlen*
DOUBLE PRECISION *y*(*)

SUBROUTINE **iCC_GIPROD(***x, xlen, y***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)

SUBROUTINE **iCC_GSPROD(***x, xlen, y***)**
SINGLE PRECISION *x*(*)
INTEGER *xlen*
SINGLE PRECISION *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the product of every node's *x*.

This operation is functionally equivalent to Intel's **g*x*prod()/G*X*PROD()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*prodto1()
iCC_G*x*PRODTO1()

Global vector PRODUCT operation with result on one node.

Synopsis

void **iCC_gdprodto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_giprodto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gsprodto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

SUBROUTINE **iCC_GDPRODTO1(***x, xlen, y, iroot***)**
DOUBLE PRECISION *x*(*)
INTEGER *xlen*
DOUBLE PRECISION *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GIPRODTO1(***x, xlen, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GSPRODTO1(***x, xlen, y, iroot***)**
SINGLE PRECISION *x*(*)
INTEGER *xlen*
SINGLE PRECISION *y*(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers, single-precision floats, or double-precision floats) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the product of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*sum()
iCC_G*x*SUM()

Global vector SUM operation with result on all nodes.

Synopsis

void **iCC_gdsum(**
    double *x*[],
    long *xlen*,
    double *y*[]**)**

void **iCC_gisum(**
    long *x*[],
    long *xlen*,
    long *y*[]**)**

void **iCC_gssum(**
    float *x*[],
    long *xlen*,
    float *y*[]**)**

SUBROUTINE **iCC_GDSUM(***x, xlen, y***)**
    DOUBLE PRECISION *x*(*)
    INTEGER *xlen*
    DOUBLE PRECISION *y*(*)

SUBROUTINE **iCC_GISUM(***x, xlen, y***)**
    INTEGER *x*(*)
    INTEGER *xlen*
    INTEGER *y*(*)

SUBROUTINE **iCC_GSSUM(***x, xlen, y***)**
    SINGLE PRECISION *x*(*)
    INTEGER *xlen*
    SINGLE PRECISION *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the sum of every node's *x*.

This operation is functionally equivalent to Intel's **g*x*sum()/G*X*SUM()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g*x*sumto1()
iCC_G*x*SUMTO1()

Global vector SUM operation with result on one node.

Synopsis

void **iCC_gdsumto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gisumto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

void **iCC_gssumto1(**
double *x*[],
long *xlen*,
double *y*[],
long *iroot***)**

SUBROUTINE **iCC_GDSUMTO1(***x, xlen, y, iroot***)**
DOUBLE PRECISION *x*(*)
INTEGER *xlen*
DOUBLE PRECISION *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GISUMTO1(***x, xlen, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)
INTEGER *iroot*

SUBROUTINE **iCC_GSSUMTO1(***x, xlen, y, iroot***)**
SINGLE PRECISION *x*(*)
INTEGER *xlen*
SINGLE PRECISION *y*(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers, single-precision floats, or double-precision floats) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the sum of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_ggath()
iCC_GGATH()

Global gather operation for contributions of unknown length.

Synopsis

void **iCC_ggath(**
char *x*[],
long *xlen*,
char *y*[],
long *ylen*,
long **ncnt*,
long *iroot***)**

SUBROUTINE **iCC_GGATH(***x, xlen, y, ylen, ncnt, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)
INTEGER *ylen*
INTEGER *ncnt*
INTEGER *iroot*

Parameters

x	Pointer to the buffer containing the node's contribution.
xlen	Size (in bytes) of the node's contribution.
y	Pointer to the buffer containing the gathered result. (Significant only on the root node.)
ylen	Size (in bytes) of the root node's *y*.
ncnt	Number of bytes actually gathered into *y*.
iroot	Node ID of the root.

Discussion

At the end of the operation, the root node's *y* contains the concatenation of all the nodes' contributions. This routine is slightly less efficient than **iCC_ggathx()/iCC_GGATHX()**.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_ggathx()
iCC_GGATHX()

Global gather operation for contributions of known length.

Synopsis

void **iCC_ggathx(**
char *x*[],
long *xlens*[],
char *y*[],
long *iroot***)**

SUBROUTINE **iCC_GGATHX(***x, xlens, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlens*(*)
INTEGER *y*(*)
INTEGER *iroot*

Parameters

x	Pointer to the buffer containing the node's contribution.
xlens	Array containing the size (in bytes) of each node's contribution (indexed by node ID).
y	Pointer to the buffer containing the gathered result. (Significant only on the root node and assumed to be large enough since the size of all nodes contributions are known beforehand.)
iroot	Node ID of the root.

Discussion

At the end of the operation, the root node's *y* contains the concatenation of all the nodes' contributions. This routine is slightly more efficient than **iCC_ggath()/iCC_GGATH()**.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_giand()
iCC_GIAND()

Global vector bitwise AND operation with result on all nodes.

Synopsis

void **iCC_giand(**
long *x*[],
long *xlen*,
long *y*[])

SUBROUTINE **iCC_GIAND(***x, xlen, y***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the bitwise ANDing of every node's *x*.

This operation is functionally equivalent to Intel's **giand()/GIAND()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_giandto1()
iCC_GIANDTO1()

Global vector bitwise AND operation with result on one node.

Synopsis

void **iCC_giandto1**(
long *x*[],
long *xlen*,
long *y*[],
long *iroot*)

SUBROUTINE **iCC_GIANDTO1(***x, xlen, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER y(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the bitwise ANDing of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gior()
iCC_GIOR()

Global vector bitwise OR operation with result on all the nodes.

Synopsis

```
void iCC_gior(
        long x[],
        long xlen,
        long y[])
```

```
SUBROUTINE iCC_GIOR(x, xlen, y)
        INTEGER x(*)
        INTEGER xlen
        INTEGER y(*)
```

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the bitwise ORing of every node's *x*.

This operation is functionally equivalent to Intel's **gior()/GIOR()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_giorto1()
iCC_GIORTO1()

Global vector bitwise OR operation with result on one node.

Synopsis

```
void iCC_giorto1(
        long x[],
        long xlen,
        long y[],
        long iroot)
```

```
SUBROUTINE iCC_GIORTO1(x, xlen, y, iroot)
        INTEGER x(*)
        INTEGER xlen
        INTEGER y(*)
        INTEGER iroot
```

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the bitwise ORing of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gland()
iCC_GLAND()

Global vector logical AND operation with result on all nodes.

Synopsis

void **iCC_gland(**
long *x*[],
long *xlen*,
long *y*[])

SUBROUTINE **iCC_GLAND(***x, xlen, y***)**
LOGICAL *x*(*)
INTEGER *xlen*
LOGICAL *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the logical ANDing of every node's *x*.

This operation is functionally equivalent to Intel's **gland()/GLAND()** except that the result is stored in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_glandto1()
iCC_GLANDTO1()

Global vector logical AND operation with result on one node.

Synopsis

void **iCC_glandto1(**
long *x*[],
long *xlen*,
long *y*[],
long *iroot***)**

SUBROUTINE **iCC_GLANDTO1(***x, xlen, y, iroot***)**
LOGICAL *x*(*)
INTEGER *xlen*
LOGICAL *y*(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers/LOGICALs) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the logical ANDing of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_glor()
iCC_GLOR()

Global logical OR operation with result on all the nodes.

Synopsis

void **iCC_glor**(
long *x*[],
long *xlen*,
long *y*[])

SUBROUTINE **iCC_GLOR**(*x, xlen, y*)
LOGICAL *x*(*)
INTEGER *xlen*
LOGICAL *y*(*)

Parameters

x	Input vector for the operation.
xlen	Length (in integers) of *x*.
y	Output vector for the operation. (May be the same as *x*.)

Discussion

At the end of the operation, each node's *y* contains the logical ORing of every node's *x*.

This operation is functionally equivalent to Intel's **glor()/GLOR()** except that the result is store in *y* and there is no *work* array.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_glorto1()
iCC_GLORTO1()

Global vector logical OR operation with result on one node.

Synopsis

void **iCC_glorto1(**
long *x*[],
long *xlen*,
long *y*[],
long *iroot***)**

SUBROUTINE **iCC_GLORTO1(***x, xlen, y, iroot***)**
INTEGER *x*(*)
INTEGER *xlen*
INTEGER *y*(*)
INTEGER *iroot*

Parameters

x	Input vector for the operation.
xlen	Length (in integers/LOGICALs) of *x*.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.

Discussion

At the end of the operation, the root node's *y* contains the logical ORing of every node's *x*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gopf()
iCC_GOPF()

User-defined global vector operation with result on all nodes.

Synopsis

void **iCC_gopf(**
    long *x*[],
    long *xlen*,
    long *size*,
    long *y*[],
    long (**f*)())

SUBROUTINE **iCC_GOPF(***x, xlen, y, f***)**
    LOGICAL *x*(*)
    INTEGER *xlen*
    INTEGER *size*
    LOGICAL *y*(*)
    EXTERNAL *f*

Parameters

x	Input vector for the operation.
xlen	Length (in units of the user's data type) of *x*.
size	Size, in bytes, of a unit of the user's data type.
y	Output vector for the operation. (May be the same as *x*.)
f	User-supplied vector function of the following format:

void *f*(
    <any type> *w*[],
    <any type> *v*[],
    long **n*)

SUBROUTINE *f*(*w, v, n*)
    <any type> *w*(*)
    <any type> *v*(*)
    INTEGER *n*

where *w* is an input-only array, *v* is an input-output array (i.e., it is over-written), and *n* is the number of elements in *w* and *v*. The value of *n* must not be changed by the user function. It is passed by reference only to provide FORTRAN compatibility.

Discussion

At the end of the operation, each node's *y* contains the application of *f*() to every node's *x*.

This operation is functionally equivalent to Intel's **gopf()/GOPF()** except that the user must specify the size of the data elements, the result is stored in *y* and there is no *work* array. Also, the interface to the user-supplied function is different than that for Intel's **gopf()/GOPF()** call.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gopfto1()
iCC_GOPFTO1()

User-defined global vector operation with result on one node.

Synopsis

void **iCC_gopfto1**(
long *x*[],
long *xlen*,
long *size*,
long *y*[],
long *iroot*,
long (**f*)())

SUBROUTINE **iCC_GOPFTO1**(*x, xlen, size, y, iroot, f*)
LOGICAL *x*(*)
INTEGER *xlen*
INTEGER *size*
LOGICAL *y*(*)
INTEGER *iroot*
EXTERNAL *f*

Parameters

x	Input vector for the operation.
xlen	Length (in units of the user's data type) of *x*.
size	Size, in bytes, of a unit of the user's data type.
y	Output vector for the operation (root node only).
iroot	ID of node to receive the result.
f	User-supplied vector function of the following format:

void *f*(
<any type> *w*[],
<any type> *v*[]
long **n*)

SUBROUTINE *f*(*w, v, n*)
<any type> *w*(*)
<any type> *v*(*)
INTEGER *n*

where *w* is an input-only array, *v* is an input-output array (i.e., it is overwritten), and *n* is the number of elements in *w* and *v*. The value of *n* must not be changed by the user function. It is passed by reference only to provide FORTRAN compatibility.

Discussion

At the end of the operation, the root node's *y* contains the application of *f*() to every node's *x*.

This operation has no functional equivalent in Intel's library. Note that the interface to the user-supplied function is different than that for Intel's **gopf()/GOPF()** call.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_grid_init()
iCC_GRID_INIT()

Tell iCC the dimensions of the partition.

Synopsis

void **iCC_grid_init**(
long *nrows*,
long *ncols*)

SUBROUTINE **iCC_GRID_INIT(***nrows, ncols*)
INTEGER *nrows*
INTEGER *ncols*

Parameters

nrows	Number of rows in the partition.
ncols	Number of columns in the partition.

Discussion

This routine only applies to the Delta and Paragon systems. On the iPSC it is a stub.

The iCC library will automatically determine the size of the partition on the Delta and Paragon systems by using Intel's **mypart()** call. However, if the user would prefer to have iCC view the partition as having different logical dimensions, then **iCC_grid_init()/ iCC_GRID_INIT()** can be called to supply these dimensions. Such a call must proceed any other iCC call, and the number of nodes in the logical partition must equal the number of nodes actually in the partition.

This operation has no functional equivalent in Intel's library.

iCC_gscatx()
iCC_GSCATX()

Global scatter operation for portions of known length.

Synopsis

void **iCC_gscatx(**
    char *x*[],
    char *y*[],
    long *ylens*[],
    long *iroot***)**

SUBROUTINE **iCC_GSCATX(***x, y, ylens, iroot***)**
    INTEGER *x*(*)
    INTEGER *y*(*)
    INTEGER *ylens*(*)
    INTEGER *iroot*

Parameters

x	Pointer to the buffer containing the root node's data.
y	Pointer to the buffer to contain the node's portion.
ylens	Array containing the size (in bytes) of each node's portion.
iroot	Node ID of the root.

Discussion

At the end of the operation, the root node's *x* is divided up among all the nodes according to *ylens*.

This operation has no functional equivalent in Intel's library.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_gsync()
iCC_GSYNC()

Tight (slower) synchronization of all nodes.

Synopsis

void **iCC_gsync()**

SUBROUTINE **iCC_GSYNC()**

Discussion

This routine guarantees that all nodes have executed it before any node continues. It also guarantees that all nodes will leave the routine simultaneously. (Use the cheaper **iCC_barrier()/iCC_BARRIER()** routine for loose synchronization.)

This is a global operation. All nodes must execute this routine for it to complete.

Group Routines

Contents

This section describes the MPI-like group routines that have been added in the latest iCC library.

Global Variables

iCC_g_GROUP_EMPTY	A group containing no nodes.
iCC_g_COMM_WORLD	A communicator whose group contains every node.
iCC_g_COMM_SELF	A communicator whose group contains only the calling node.
iCC_g_COMM_NULL	A null communicator.
iCC_g_GROUP_NULL	A null group.
iCC_g_DATATYPE_NULL	A null datatype.
iCC_g_OP_NULL	A null operator.

Predefined Datatypes

C:	**iCC_g_INT, iCC_g_LONG, iCC_g_FLOAT, iCC_g_DOUBLE, iCC_g_CHAR, iCC_g_BYTE**
Fortan:	**iCC_g_INTEGER, iCC_g_REAL, iCC_g_DOUBLE_PRECISION, iCC_g_LOGICAL, iCC_g_CHARACTER, iCC_g_BYTE**

Predefined Operators

Integer	Single Precision	Double Precision	Logical	Bitwise
iCC_g_IMAX	**iCC_g_SMAX**	**iCC_g_DMAX**	**iCC_g_LAND**	**iCC_g_BAND**
iCC_g_IMIN	**iCC_g_SMIN**	**iCC_g_DMIN**	**iCC_g_LOR**	**iCC_g_BOR**
iCC_g_IPROD	**iCC_g_SPROD**	**iCC_g_DPROD**		
iCC_g_ISUM	**iCC_g_SSUM**	**iCC_g_DSUM**		

Return Codes

0	Successful return
1	Null pointer passed in as an argument
2	Null communicator passed in as an argument
3	Null group passed in as an argument
4	Empty group improperly passed in as an argument
5	Null operator passed in as an argument
6	Null datatype passed in as an argument
7	Mismatched datatype/operator passed in as arguments
8	Intercommunicator passed in as an argument (not supported)
9	Root out of range (not in group)
10	Calling node's position in group invalid
11	Calling node out of range (not in group)
12	iCC group interface has not been initialized with **iCC_g_group_init**()/ **iCC_g_GROUP_INIT()**
13	Memory allocation failure
14	Receive element counts do not equal send element counts
15	Improper block sizes passed in as arguments

iCC_g_Allgatherv()
iCC_g_ALLGATHERV()

Gathers (collects) the send buffers from all the nodes associated with the given communicator into the receive buffers on all the nodes in that group.

Synopsis

long **iCC_g_Allgatherv(**
    void **sendbuf*,
    long *sendcount*,
    iCC_g_Datatype *sendtype*,
    void **recvbuf*,
    long **recvcounts*,
    long **displs*,
    iCC_g_Datatype *recvtype*,
    iCC_g_Comm *comm***)**

SUBROUTINE **iCC_g_ALLGATHERV(***sendbuf, sendcount, sendtype, recvbuf,*
    *recvcounts, displs, recvtype, comm, ierror***)**
    <any type> *sendbuf*(*), *recvbuf*(*)
    INTEGER *sendcount, sendtype, recvcounts, displs, recvtype, comm, ierror*

Parameters

sendbuf	Pointer to the data items to be collected from the calling node.
sendcount	Number of data items to be collected from the calling node.
sendtype	iCC type of the data items to be collected from the calling node.
recvbuf	Pointer to buffer into which the data is to be collected.
recvcounts	Number of data items to be collected from each node.
displs	Displacements into the receive buffer for each node's contribution (not supported).
recvtype	iCC type of the buffer into which the data items are to be collected.
comm	Communicator under which the collect is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the receive buffer on every node in the given communicator's group will contain the contents, by rank, of the send buffers of every node in that group.

This routine differs from the MPI version in that it does not support receive buffer displacements, and in that it requires all nodes to provide *recvcounts*.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Allreduce()
iCC_g_ALLREDUCE()

Uses the specified iCC operator to reduce the send buffers of all the nodes associated with the given communicator into the receive buffer on every node in that group.

Synopsis

long **iCC_g_Allreduce(**
    void **sendbuf*,
    void **recvbuf*,
    long *count*,
    iCC_g_Datatype *datatype*,
    iCC_g_Op *op*,
    iCC_g_Comm *comm***)**

SUBROUTINE **iCC_g_ALLREDUCE(***sendbuf, recvbuf, count, datatype, op, comm,*
    *ierror***)**
    <any type> *sendbuf*(*), *recvbuf*(*)
    INTEGER *count, datatype, op, comm, ierror*

Parameters

sendbuf	Pointer to the data items to be reduced.
recvbuf	Pointer to the buffer into which the data is to be reduced.
count	Number of data items to be reduced on each node.
datatype	iCC data type of the items to be reduced.
op	iCC operation to be used in the reduction.
comm	Communicator under which the reduction is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the receive buffer on every node in the given communicator's group will contain the reduction of the send buffers of all of the nodes in that group.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Barrier()
iCC_g_BARRIER()

Loose (faster) synchronization of all nodes associated with the given communicator.

Synopsis

long **iCC_g_Barrier(**
    iCC_g_Comm *comm***)**

SUBROUTINE **iCC_g_BARRIER(***comm, ierror***)**
    INTEGER *comm, ierror*

Parameters

comm	Communicator on which the barrier is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

This routine guarantees that all nodes in the group associated with the given communicator have executed it before any node in that group continues. It does not guarantee that all nodes will leave the routine at the same time. (Use the more expensive **iCC_g_Sync()/ iCC_g_SYNC()** routine to insure that all nodes exit the routine simultaneously.)

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Bcast()
iCC_g_BCAST()

Broadcast buffer from the specified node to all nodes associated with the given communicator.

Synopsis

long **iCC_g_Bcast**(
    void **buffer*,
    long *count*,
    iCC_g_Datatype *datatype*,
    long *root*,
    iCC_g_Comm *comm*)

SUBROUTINE **iCC_g_BCAST(***buffer, count, datatype, root, comm, ierror***)**
    <any type> *buffer*(*)
    INTEGER *count, datatype, root, comm, ierror*

Parameters

buffer	Pointer to data items to be broadcast.
count	Number of data items to be broadcast.
datatype	iCC type of the data items to be broadcast.
root	Rank of the node within the given communicator's group to perform the broadcast.
comm	Communicator under which the broadcast is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the contents of the buffer on the root node will be replicated in the buffers of every node in the given communicator's group.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Comm_create()
iCC_g_COMM_CREATE()

Create a new communicator formed from a subgroup of the group associated with the given communicator.

Synopsis

long **iCC_g_Comm_create(**
    iCC_g_Comm *comm,*
    iCC_g_Group *group,*
    iCC_g_Comm **newcomm***)**

SUBROUTINE **iCC_g_COMM_CREATE(***comm, group, newcomm, ierror***)**
    INTEGER *comm, group, newcomm, ierror*

Parameters

comm	Communicator from which the new communicator is derived.
group	Subgroup of the given communicator's group from which to form the new communicator.
newcomm	Communicator created from the subgroup of the given communicator.
ierror	Error returned (FORTRAN version only).

Discussion

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Comm_dup()
iCC_g_COMM_DUP()

Creates a duplicate of the given communicator.

Synopsis

long **iCC_g_Comm_dup(**
    iCC_g_Comm *comm,*
    iCC_g_Comm **newcomm***)**

SUBROUTINE **iCC_g_COMM_DUP(***comm, newcomm, ierror***)**
    INTEGER *comm, newcomm, ierror*

Parameters

comm	Communicator to be duplicated.
newcomm	Duplicate of given comminicator.
ierror	Error returned (FORTRAN version only).

Discussion

None.

iCC_g_Comm_free()
iCC_g_COMM_FREE()

Marks a communicator for deallocation.

Synopsis

long **iCC_g_Comm_free(**
    iCC_g_Comm **comm*)

SUBROUTINE **iCC_g_COMM_FREE(***comm, ierror***)**
    INTEGER *comm, ierror*

Parameters

comm	Communicator to be marked for deallocation.
ierror	Error returned (FORTRAN version only).

Discussion

The communicator will not actually be deallocated until its reference counter is equal to zero.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Comm_grid()
iCC_g_COMM_GRID()

Returns the logical grid and row/column for the calling node that iCC has determined for the group associated with the given communicator.

Synopsis

long **iCC_g_Comm_grid(**
    iCC_g_Comm *comm,*
    long **nrow,*
    long **ncol,*
    long **myrow,*
    long **mycol***)**

SUBROUTINE **iCC_g_COMM_GRID(***comm, nrow, ncol, myrow, mycol, ierror***)**
    INTEGER *comm, nrow, ncol, myrow, mycol, ierror*

Parameters

comm	Communicator for which to return iCC's logical grid.
nrow	Number of rows in the logical grid.
ncol	Number of columns in the logical grid.
myrow	Row of the calling node in the logical grid.
mycol	Column of the calling node in the logical grid.
ierror	Error returned (FORTRAN version only).

Discussion

This routine is not provided by MPI. It is specific to iCC on mesh architectures.

iCC_g_Comm_group()
iCC_g_COMM_GROUP()

Returns the group handle associated with the given communicator.

Synopsis

long **iCC_g_Comm_group(**
    iCC_g_Comm *comm,*
    iCC_g_Group **group*)

SUBROUTINE **iCC_g_COMM_GROUP(***comm, group, ierror***)**
    INTEGER *comm, group, ierror*

Parameters

comm	Communicator for which the associated group is returned.
group	Group associated with the given communicator.
ierror	Error returned (FORTRAN version only).

Discussion

The group returned is separately allocated from the internal group of the communicator; it must be explicitly deallocated with **iCC_g_Group_free()/iCC_g_GROUP_FREE()**.

iCC_g_Comm_rank()
iCC_g_COMM_RANK()

Returns the rank of the calling node within the group associated with the given communicator.

Synopsis

long **iCC_g_Comm_rank(**
iCC_g_Comm *comm,*
long **rank*)

SUBROUTINE **iCC_g_COMM_RANK(***comm, rank, ierror***)**
INTEGER *comm, rank, ierror*

Parameters

comm	Communicator for which the rank of the calling node is to be determined.
rank	Rank of the calling node within the group associated with the given communicator.
ierror	Error returned (FORTRAN version only).

Discussion

None.

iCC_g_Comm_size()
iCC_g_COMM_SIZE()

Returns the size of the group associated with the given communicator.

Synopsis

long **iCC_g_Comm_size(**
    iCC_g_Comm *comm,*
    long **size*)

SUBROUTINE **iCC_g_COMM_SIZE(***comm, size, ierror***)**
    INTEGER *comm, size, ierror*

Parameters

comm	Communicator whose group's size is returned.
size	Size of the group associated with the given communicator.
ierror	Error returned (FORTRAN version only).

Discussion

None.

iCC_g_debug()
iCC_g_DEBUG()

Set the debug level for iCC group routines.

Synopsis

void **iCC_g_Debug(**
    int *level***)**

SUBROUTINE **iCC_g_DEBUG(***level***)**
    INTEGER *level*

Parameters

level	Debug level to which iCC group routines are set.

Discussion

This routine causes the iCC library to print warnings and other debugging information at varying levels of detail. The default (level = 0) is debugging off. More debugging information is printed for higher levels (1,2,3,...).

iCC_g_Gatherv()
iCC_g_GATHERV()

Gathers the send buffers from all the nodes assocated with the given communicator into the receive buffer on the specified node.

Synopsis

long **iCC_g_Gatherv**(
void **sendbuf*,
long *sendcount*,
iCC_g_Datatype *sendtype*,
void **recvbuf*,
long **recvcounts*,
long **displs*,
iCC_g_Datatype *recvtype*,
long *root*,
iCC_g_Comm *comm*)

SUBROUTINE **iCC_g_GATHERV(***sendbuf, sendcount, sendtype, recvbuf, recvcounts,*
*displs, recvtype, root, comm, ierror***)**
<any type> *sendbuf*(*), *recvbuf*(*)
INTEGER *recvcounts*(*), *displs*(*)
INTEGER *sendcount, sendtype, recvtype, root, comm, ierror*

Parameters

sendbuf	Pointer to the data items to be gathered from the calling node.
sendcount	Number of data items to be gathered from the calling node.
sendtype	iCC type of the data items to be gathered from the calling node.
recvbuf	Pointer to buffer into which the data is to be gathered (significant only on the root).
recvcounts	Number of data items (by rank) to be gathered from each node.
displs	Displacements into the receive buffer for each node's contribution (not supported).
recvtype	iCC type of the buffer into which data items are to be gathered.
root	Rank of the node within the given communicator's group to be gathered to.
comm	Communicator under which the gather is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the receive buffer on the root node will contain the contents, by rank, of the send buffers of every node in the given communicator's group.

This routine differs from the MPI version in that it does not support receive buffer displacements, and in that it requires all nodes to provide *recvcounts*.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Group_block_incl()
iCC_g_GROUP_BLOCK_INCL()

Creates a subgroup of nodes specified by rows and columns with the given parent group.

Synopsis

long **iCC_g_Group_block_incl(**
iCC_g_Group *group,*
long *startrow,*
long *startcol,*
long *nrow,*
long *rowstride,*
long *ncol,*
long *colstride,*
iCC_g_Group **newgroup*)

SUBROUTINE **iCC_g_GROUP_BLOCK_INCL(***group, startrow, startcol, nrow, row stride, ncol, colstride, newgroup, ierror***)**
INTEGER *group, startrow, startcol, nrow, rowstride, ncol, colstride, newgroup, ierror*

Parameters

group	Group from which to extract the new group.
startrow	Topmost row for new group block.
startcol	Leftmost column for new group block.
nrow	Number of rows to extract.
rowstride	Stride from one row to the next.
ncol	Number of columns to extract.
colstride	Stride from one column to the next.
newgroup	Subgroup derived from the parent group.
ierror	Error returned (FORTRAN version only).

Discussion

The group must be explicitly deallocated with **iCC_g_Group_free**()/**iCC_g_GROUP_FREE**(), even if it is incorporated into a communicator using **iCC_g_Comm_create**()/**iCC_g_COMM_CREATE**().

This routine is not provided by MPI. It is specific to iCC on mesh architectures.

iCC_g_Group_finalize()
iCC_g_GROUP_FINALIZE()

Cleanup and shut down iCC group capabilities.

Synopsis

long **iCC_g_Group_finalize**()

SUBROUTINE **iCC_g_GROUP_FINALIZE(***ierror***)**
    INTEGER *ierror*

Parameters

ierror	Error returned (FORTRAN version only).

Discussion

This routine must be called to ensure a clean shutdown of the iCC group facilities.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g_Group_free()
iCC_g_GROUP_FREE()

Marks a group for deallocation.

Synopsis

long **iCC_g_Group_free(**
    iCC_g_Group **group*)

SUBROUTINE **iCC_g_GROUP_FREE(***group, ierror***)**
    INTEGER *group, ierror*

Parameters

group	Group to be marked for deallocation.
ierror	Error returned (FORTRAN version only).

Discussion

The group will not actually be deallocated until its reference counter is equal to zero.

iCC_g_Group_grid()
iCC_g_GROUP_GRID()

Returns the logical grid and row/column for the calling node that iCC has determined for the given group.

Synopsis

long **iCC_g_Group_grid(**
    iCC_g_Group *group,*
    long **nrow,*
    long **ncol,*
    long **myrow,*
    long **mycol***)**

SUBROUTINE **iCC_g_GROUP_GRID(***group, nrow, ncol, myrow, mycol, ierror***)**
    INTEGER *group, nrow, ncol, myrow, mycol, ierror*

Parameters

group	Group for which to return iCC's logical grid.
nrow	Number of rows in the logical grid.
ncol	Number of columns in the logical grid.
myrow	Row of the calling node in the logical grid.
mycol	Column of the calling node in the logical grid.
ierror	Error returned (FORTRAN version only).

Discussion

This routine is not provided by MPI. It is specific to iCC on mesh architectures.

iCC_g_Group_incl()
iCC_g_GROUP_INCL()

Creates a subgroup of processes specified by ranks with the given parent group.

Synopsis

long **iCC_g_Group_incl(**
    iCC_g_Group *group,*
    long *n,*
    long **ranks,*
    iCC_g_Group **newgroup***)**

SUBROUTINE **iCC_g_GROUP_INCL(***group, n, ranks, newgroup, ierror***)**
    INTEGER *ranks*(*)
    INTEGER *group, n, newgroup, ierror*

Parameters

group	Group from which to extract the new group.
n	Number of processes in the new group.
ranks	Ranks of nodes within the parent group to include in the new group.
newgroup	Subgroup derived from the parent group.
ierror	Error returned (FORTRAN version only).

Discussion

If the ranks provided happen to form a subgrid of the physical mesh, iCC will operate on the new group as a 2-D mesh; otherwise, it will be treated as a linear array. The group must be explicitly deallocated with **iCC_g_Group_free()/iCC_g_GROUP_FREE()**, even if it is incorporated into a communicator using **iCC_g_Comm_create()/ iCC_g_COMM_CREATE()**.

iCC_g_Group_info()
iCC_g_GROUP_INFO()

Prints useful information about the given group.

Synopsis

long **iCC_g_Group_info(**
FILE **fp***,**
iCC_g_Group *group***)**

SUBROUTINE **iCC_g_GROUP_INFO(***group***)**
INTEGER *group*

Parameters

fp	File to which the information should be printed.
group	Group for which to print information.

Discussion

The FORTRAN version will print to stardard output, while the C version allows the user to specify an output file.

iCC_g_Group_init()
iCC_g_GROUP_INIT()

Initialize the iCC group capabilities.

Synopsis

long **iCC_g_Group_init**()

SUBROUTINE **iCC_g_GROUP_INIT(***ierror***)**
    INTEGER *ierror*

Parameters

ierror	Error returned (FORTRAN version only).

Discussion

This routine must be called before any other iCC group routines are called.

This is a global operation. All nodes must execute this routine for it to complete.

iCC_g_Group_rank()
iCC_g_GROUP_RANK()

Returns calling node's rank in the given group.

Synopsis

long **iCC_g_Group_rank(**
    iCC_g_Group *group,*
    long **rank***)**

SUBROUTINE **iCC_g_GROUP_RANK(***group, rank, ierror***)**
    INTEGER *group, rank, ierror*

Parameters

group	Group in which the calling node's rank is to be determined.
rank	Calling node's rank (or position) in the given group.
ierror	Error returned (FORTRAN version only).

Discussion

None.

iCC_g_Group_size()
iCC_g_GROUP_SIZE()

Returns the number of nodes in given group.

Synopsis

long **iCC_g_Group_size(**
    iCC_g_Group *group,*
    long **size*)

SUBROUTINE **iCC_g_GROUP_SIZE(***group, size, ierror***)**
    INTEGER *group, size, ierror*

Parameters

group	Group whose size is to be determined.
size	Number of nodes in the given group.
ierror	Error returned (FORTRAN version only).

Discussion

None.

iCC_g_Op_create()
iCC_g_OP_CREATE()

Binds an iCC operator to the given function.

Synopsis

long **iCC_g_Op_create**(
    iCC_g_Uop **uop*,
    iCC_g_Datatype *datatype*,
    long *commute*,
    iCC_g_Op *op*)

SUBROUTINE **iCC_g_OP_CREATE**(*uop, datatype, commute, op, ierror*)
    EXTERNAL *uop*
    LOGICAL *commute*
    INTEGER *datatype, op, ierror*

Parameters

uop User function of the form:

void *uop*(
    <anytype> *x*[],
    <anytype> *y*[],
    long **n*)

SUBROUTINE *uop*(*x, y, n*)
    <anytype> *x*(*)
    <anytype> *y*(*)
    INTEGER *n*

datatype iCC data type of the items the operator is to be applied to.
commute Whether or not the operator is commutative (only commutative operators are supported).
op iCC operator to be associated with the user function.
ierror Error returned (FORTRAN version only).

Discussion

This routine allows the user to create new iCC operators to be used by routines such as **iCC_g_Reduce()/iCC_g_REDUCE()**. The user operator takes two data vectors, x and y, both of length n, and performs the operation: $y \leftarrow x$ op y

This calling sequence for the user operation differs from the MPI standard in that it does not include a data type parameter.

iCC_g_Op_free()
iCC_g_OP_FREE()

Deallocates an iCC operator.

Synopsis

long **iCC_g_Op_free**(
    iCC_g_Op **op*)

SUBROUTINE **iCC_g_OP_FREE(***op, ierror***)**
    INTEGER *op, ierror*

Parameters

op	iCC operator to be deallocated.
ierror	Error returned (FORTRAN version only).

Discussion

iCC operators can (and should) be deallocated if they are no longer to be used.

iCC_g_Reduce()
iCC_g_REDUCE()

Uses the specified iCC operator to reduce the send buffers of all the nodes assocated with the given communicator into the receive buffer on the root node of that group.

Synopsis

long **iCC_g_Reduce**(
    void **sendbuf*,
    void **recvbuf*,
    long *count*,
    iCC_g_Datatype *datatype*,
    iCC_g_Op *op*,
    long *root*,
    iCC_g_Comm *comm*)

SUBROUTINE **iCC_g_REDUCE(***sendbuf, recvbuf, count, datatype, op, root, comm, ierror***)**
    <any type> *sendbuf*(*), *recvbuf*(*)
    INTEGER *count, datatype, op, root, comm, ierror*

Parameters

sendbuf	Pointer to the data items to be reduced onto the root node.
recvbuf	Pointer to buffer into which the data is to be reduced (significant only on the root).
count	Number of data items to be reduced on each node.
datatype	iCC data type of the items to be reduced.
op	iCC operation to be used in the reduction.
root	Rank of the node within the given communicator's group to be reduced to.
comm	Communicator under which the reduction is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the receive buffer on the root node will contain the reduction of the send buffers of all of the nodes in the given communicator's group.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Reduce_scatter()
iCC_g_REDUCE_SCATTER()

Uses the specified iCC operator to reduce the send buffers of all the nodes assocated with the given communicator, leaving the result distributed in the receive buffers of the nodes in that group.

Synopsis

long **iCC_g_Reduce_scatter(**
    void **sendbuf*,
    void **recvbuf*,
    long **recvcounts*,
    iCC_g_Datatype *datatype*,
    iCC_g_Op *op*,
    iCC_g_Comm *comm***)**

SUBROUTINE **iCC_g_REDUCE_SCATTER(***sendbuf, recvbuf, recvcounts, datatype,*
    *op, comm, ierror***)**
    <any type> *sendbuf*(*), *recvbuf*(*)
    INTEGER *recvcounts*(*)
    INTEGER *datatype, op, comm, ierror*

Parameters

sendbuf	Pointer to the data items to be reduced.
recvbuf	Pointer to the buffer into which a portion of the data is to be reduced.
recvcounts	Number of data items (by rank) to be reduced on each node.
datatype	iCC data type of the items to be reduced.
op	iCC operation to be used in the reduction.
comm	Communicator under which the reduction is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the receive buffer on every node in the given communicator's group will contain a portion of the reduction of the send buffers of all of the nodes in that group.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Scatterv()
iCC_g_SCATTERV()

Scatters portions of the send buffer on the root node into the receive buffers of all the nodes assocated with the given communicator.

Synopsis

long **iCC_g_Scatterv**(
    void **sendbuf*,
    long **sendcounts*,
    long **displs*,
    iCC_g_Datatype *sendtype*,
    void **recvbuf*,
    long *recvcount*,
    iCC_g_Datatype *recvtype*,
    long *root*,
    iCC_g_Comm *comm*)

SUBROUTINE **iCC_g_SCATTERV(***sendbuf, sendcounts, displs, sendtype, recvbuf,*
    *recvcount, recvtype, root, comm, ierror***)**
    <any type> *sendbuf*(*), *recvbuf*(*)
    INTEGER *sendcounts*(*), *displs*(*)
    INTEGER *sendtype, recvcount, recvtype, root, comm, ierror*

Parameters

sendbuf	Pointer to the data items to be scattered from the root node (significant only on the root).
sendcounts	Number of data items (by rank) to be scattered to each node.
displs	Displacements into send buffer for each node's portion (not supported).
sendtype	iCC type of the data items to be scattered from root node.
recvbuf	Pointer to the buffer into which data is to be scattered.
recvcount	Number of data items to be scattered to the calling node.
recvtype	iCC type of the buffer into which data items are to be scattered.
root	Rank of the node within the given communicator's group to be scattered from.
comm	Communicator under which the scatter is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

At the end of the operation, the receive buffers on each node in the given communicator's group will contain a portion, by rank, of the send buffer on the root node of that group.

This routine differs from the MPI version in that it does not support send buffer displacements, and in that it requires all nodes to provide *sendcounts*.

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.

iCC_g_Sync()
iCC_g_SYNC()

Tight (slower) synchronization of all nodes associated with the given communicator.

Synopsis

long **iCC_g_Sync(**
    iCC_g_Comm *comm***)**

SUBROUTINE **iCC_g_SYNC(***comm, ierror***)**
    INTEGER *comm, ierror*

Parameters

comm	Communicator on which the synchronization is to be performed.
ierror	Error returned (FORTRAN version only).

Discussion

This routine guarantees that all nodes in the group associated with the given communicator have executed it before any node continues and roughly guarantees that all nodes will leave the routine at the same time. (Use the less expensive **iCC_g_Barrier()/ iCC_g_BARRIER()** routine if such tight synchronization is unnecessary.)

This is a group operation. All nodes in the given communicator's group must execute this routine for it to complete.