intel

iPSC™ Technical Report

No. 2
Communication Utilities for the iPSC™

Communication Utilities For The iPSC™

Cleve Moler

David S. Scott

Intel Scientific Computers

August 1986

ABSTRACT

A library of communication subroutines for the iPSC is described which provides facilities for global message broadcast, global sums and other commutative arithmetic operations, embedding of meshes into the hypercube, and very long messages. The descriptions correspond to the FORTRAN versions of the codes. Versions written in C are also available.

1. INTRODUCTION

The communication primitives in the basic iPSC operating system provide for sending a message from one single process to another. We have found it useful in applications programming to expand these facilities to provide some higher level constructs for interprocess communication. Included are:

- Global Broadcast -- send a message from one node to all other nodes.
- Global Operations -- commutative arithmetic operations, such as sum or maximize, with contributions from each node.
- Global Concatenation -- collect messages from each node.
- Meshes -- provide each node with left - right, or north - east - south - west, neighbors.
- Long messages -- handle messages longer than 16K bytes.
- Sparse Global Send -- Send and receive with selected subset of nodes.

The iPSC system send and receive calls use both a node number and a process id to specify the destination of a message. This allows multiprocessing of the nodes. However, the utilities described here assume only one participating process on each node. For simplicity it is assumed that each participating process has the same PID. For this reason the PID is not included in the calls. It is obtained internally by a call to MYPID().

---

iPSC™ is a trademark of Intel Corporation

Several of the subroutines allow operations involving set of nodes called subcubes. By a subcube of dimension d we mean the set of $p = 2^d$ nodes whose addresses differ in their low order d bits. For example, the subcube of dimension 3 containing node 13 consists of nodes 8 through 15. Note that such subcubes correspond to neighboring sets of boards in the iPSC hardware; a 5 dimensional subcube is a full 32 node iPSC system (or cabinet), a 3 dimensional subcube is either the left or the right half of a horizontal row of boards, and so on.

An example problem which uses some of the subroutines is given after all of the subroutine descriptions.

2. GLOBAL BROADCAST

This section describes a procedure for one node (called the root) to distribute the same message to all the other nodes. It is also possible to limit the broadcast to a lower order subcube, or to initiate the broadcast from the host. It is assumed that each participating process has the same PID and that the host knows this common PID if it is the root. The receiving processes call GRECVW while the root calls GSENDW. If the host is the root, it just uses SENDMSG to send the message to any one of the receiving processes.

The calling sequences are:

```
SUBROUTINE GSENDW (CI, TYPE, BUF, LEN, DIM)
INTEGER CI, TYPE, LEN, DIM
BYTES BUF(*)
```

and

```
SUBROUTINE GRECVW (CI, TYPE, BUF, LEN, CNT, DIM)
INTEGER CI, TYPE, LEN, CNT, DIM
BYTES BUF(*)
```

The parameters are

CI	The channel number to be used (already opened).
TYPE	The message type (must be the same for all calls).
BUF	The message buffer. The type of BUF is irrelevant.
LEN	The length of the message in bytes for sending. The length of the buffer in bytes for receiving.
CNT	Returned as the length (in bytes) of the message received NOTE: CNT > LEN is an error return condition since only LEN bytes will be stored in BUF and the rest of the message will be lost. In this case an error message will be syslogged but the remnants of the message will be sent on.
DIM	The dimension of the subcube participating in the send NOTE: only those nodes which will participate in the send can call GRECVW. Any other node which calls it will never return.

The implementation of GSENDW and GRECVW involves a subcube spanning tree, rooted at the root node. This is also known as "induction on the cube dimension." GSENDW sends the message from the root to all of its nearest neighbors. In GRECVW, if a node receives the message from a neighbor, it then sends it on to some, but not all, of the remaining neighbors. If it receives it from the host, it sends it on to all of its neighbors. The hypercube neighbors of a node are those nodes whose addresses differ from the

given address in exactly one bit. The list of neighbors can be ordered by increasing bit number. In [1] it was shown that each node should send the data to all neighbors on this list after the node they receive it from. In particular, it is not necessary for the receiving nodes to know the root of the broadcast.

It is likely that future versions of the iPSC node operating system will offer global broadcast as a primitive operation. Calling SEND or SENDW with NID = -1 will initiate the broadcast of a message to all the nodes in the cube. Other negative destination node numbers will initiate broadcasts to subcubes. The message will be received with an RECV or RECVW and the operating system will automatically manage the spanning tree.

3. GLOBAL COMMUTATIVE OPERATIONS

Sometimes all the nodes have some data that must be accumulated. The most common example is computing a distributed innerproduct. In these situations an inverse or dual of the global send is needed. All the participants call the subroutine GOP. Four different operations are included in the distributed version: addition, multiplication, maximum, and minimum. It is easy to modify the source code to include other operations. If the operation is applied to a vector, then it is applied component-wise to the elements. The implementation allows accumulation over a lower order subcube. If root is specified as the host, then the message is accumulated in the lowest numbered node and then sent on to the host (with the same TYPE and same PID). The calling sequence is:

```
SUBROUTINE GOP (CI, TYPE, X, N, OP, ROOT, DIM, WORK)
INTEGER CI, TYPE, N, ROOT, DIM
CHARACTER*1 OP
DOUBLE PRECISION X(N), WORK(N)
```

Input

CI — Channel number (previously opened)

TYPE — Message type. Must be the same for all participating processes. There must be no other messages of this type in the system.

X — The input vector to be used in the operation.

N — The length of the vectors.

OP — '+' sum
'*' product
'M' maximum
'm' minimum

ROOT — Node number of root process. (-32768 for the host)

DIM — The size of the subcube participating.

On output

X — For the root process, X contains the desired result. For all other processes, X contains the partial result for their subtrees.

Workspace

WORK — Used to receive other contributions.

4. GLOBAL CONCATENATION

Closely related to the global commutative operation is the need to collect a distributed vector in one processor. The subroutine provided for this purpose is GCAT, which globally concatenates a vector. It is possible to concatenate over a lower order subcube.

The actual type of BUF is irrelevent, it is concatenated by byte. If the root is zero, then the contributions are in ascending order (ie. nodes 0, 1, 2, 3,...,). If any other node is root, then the order obtained is the exclusive-or of the ascending order with the root. If the root is the host (-32768), the lowest numbered node acts as the root, but then sends the message on to the host (with the same type and pid). Many participating nodes receive contributions from several children. The resulting concatenation is sensitive to the order in which these contributions are received. To insure that the contributions are received in order it is necessary to use type numbers in addition to the specified number TYPE. In particular all the type numbers TYPE, TYPE+1, TYPE+2, ... TYPE+DIM-1 are used.

The calling sequence is:

```
SUBROUTINE GCAT (CI, TYPE, BUF, BUFLEN, MSGLEN, CNT, ROOT, DIM)
INTEGER CI, TYPE, BUFLEN, MSGLEN, CNT, ROOT, DIM
BYTES BUF(*)
```

Input:

CI	Channel number (previously opened).
TYPE	Message type. Must be the same for all participating processes. There must be no other messages of this type or of types TYPE+1, TYPE+2, ..., TYPE+DIM-1 in the system.
BUF	The input vector to be concatenated. The actual type of BUF is irrelevent.
BUFLEN	The length of BUF (in bytes).
MSGLEN	The length of this contribution (in bytes).
ROOT	Node id of root process (which will get the full message).
DIM	The size of the subcube participating.

Output:

BUF	For the root process, BUF contains the desired result. For all other processes, BUF contains the partial result for their subtrees.
CNT	The number of meaningful bytes in BUF.

5. EMBEDDING MESHES

Many applications require computations on a rectangular mesh. It is natural to subdivide the mesh into large rectangles and assign one processor to each rectangle. If rectangles only need to communicate with adjacent rectangles, it is desirable to map adjacent rectangles onto nearest neighbors in the hypercube. MESH computes an embedding of a rectangular mesh of nodes into the hypercube that preserves nearest neighbor connections. The subroutine takes the dimensions of the mesh, the boundary conditions for the mesh, and the node number of interest and computes the coordinates of the node and the node numbers of its nearest neighbors in each coordinate direction. An node number of -1 is used as NIL to indicate a

nonexistent neighbor for a boundary node. If the given node number is not used in the specified mesh, then all of the output parameters are set to -1.

The calling sequence is

```
SUBROUTINE MESH (ME, DIM, LINDIM, PERBC, COORDS, PRED, SUCC, WORK)
INTEGER ME, DIM, LINDIM(DIM), PERBC(DIM), COORDS(DIM),
  PRED(DIM), SUCC(DIM), WORK(DIM)
```

where

Input:

ME	The node number of interest.
DIM	The physical dimension of the mesh (usually 1, 2, or 3).
LINDIM	The linear dimension (number of processors) in each direction of the mesh.
PERBC	A flag indicating whether a particular direction has periodic boundary conditions. PERBC should be set as follows: PERBC(I) = 0 for non periodic (closed) boundaries PERBC(I) = 1 for periodic boundaries

Output:

COORDS	The coordinates of ME in the mesh, starting with 0, so that the largest the I-th coordinate can be is LINDIM(I)-1. If ME is not part of the mesh all coordinates are returned as -1.
PRED	PRED(I) is the node number proceding ME in the I-th direction. -1 is used when there is no predecessor.
SUCC	SUCC(I) is the node number succeeding ME in the I-th direction. -1 is used when there is no successor.

Workspace

WORK	holds the result of rounding the elements of LINDIM up to the next power of two.

If messages must be sent diagonally in the mesh, then the diagonal neighbors can be identified by calls to mesh with the neighbor's node numbers.

Note that if the mesh is too large to fit in the hypercube, MESH will send an error message to the system log, but it will then return the correct (but perhaps nonexistent) neighboring node numbers.

6. LONG MESSAGES

The messages processed by the primitives SEND and RECV are limited to a maximum of 16K bytes. Three subroutines are provided to allow node-node and host-node transmission of long messages. A node-host utility is impossible since messages cannot be filtered by type on the host.

The three calling sequences are

For sending from a node:

```
SUBROUTINE LSENDW (CI, TYPE, BUF, LEN4, NODE, PID)
INTEGER CI, TYPE, NODE, PID
INTEGER*4 LEN4
BYTES BUF(LEN4)
```

For sending from the host:

```
SUBROUTINE LSENDMSG (CI, TYPE, BUF, LEN4, NODE, PID)
INTEGER CI, TYPE, NODE, PID
INTEGER*4 LEN4
BYTES BUF(LEN4)
```

For receiving (on a node):

```
SUBROUTINE LRECVW (CI, TYPE, BUF, LEN4, CNT4, NODE, PID)
INTEGER CI, TYPE, NODE, PID
INTEGER*4 LEN4, CNT4
BYTES BUF(LEN4)
```

Where

CI — A channel number (previously opened).

TYPE — The type of the message.

BUF — The message buffer (which may actually be any type).

LEN4 — INTEGER*4. The length of the message in bytes for sending. The length of BUF in bytes for receiving.

CNT4 — INTEGER*4. The length of the received message.

NODE — The source node.

PID — The source pid.

The subroutines send the message in 16K byte chunks and terminate with a final chunk which is smaller than 16K bytes. If the message is exactly a multiple of 16K bytes, then a zero length message is sent, however the receiving buffer must be large enough to receive one more byte (to prevent an illegal subscript when the receive is called).

7. SPARSE GLOBAL SEND

In some applications a specific node may have a message which must be sent to a specified subset of the other nodes. Two subroutines are provided for this purpose. SPGSNW initiates a sparse send and all receiving nodes must call SPGRCW. The sending node specifies the target nodes by a list of integers. The calling sequences are

For sending:

```
SUBROUTINE SPGSNW (CI, TYPE, BUF, LEN, LIST, NUMBER)
INTEGER CI, TYPE, LEN, LIST(*), NUMBER
BYTES BUF(*)
```

For receiving:

```
SUBROUTINE SPGRCW (CI, TYPE, BUF, LEN, CNT)
INTEGER CI, TYPE, LEN, CNT
BYTES BUF(*)
```

The parameters are

CI An open channel.

TYPE A legal message type (must be the same for all participating calls)

BUF The buffer.

LEN For sending: the length of the message in bytes.
For receiving: the length of BUF.
Note: SEE RESTRICTIONS BELOW

CNT The length of the received message in bytes.

LIST The list of node numbers which should receive the data. The list is destroyed.

NUMBER The number of node numbers in list.

Restrictions on use:

SPGSNW appends a list of integer targets to the actual message. Thus the sending buffer BUF must be long enough to hold these integers. It is sufficient if the length of the sending buffer is at least

LEN + INTSIZE*NUMBER

where INTSIZE is the number of bytes per integer.

SPGRCW receives this appended message and so the LEN of the receiving buffer must be at least

CNT + INTSIZE*NUMBER.

Finally, to simplify the appending process and avoid possible alignment problems, the length of the message is implicitly rounded up to a multiple of INTSIZE. After the integers are stripped off this rounded length is returned as CNT. Therefore to avoid the accidental inclusion of garbage bytes, the sending LEN should also be a multiple of INTSIZE.

If the initiating node is called the **root**, and all the other participating nodes are called **targets**, then the sparse global send does not just send from the root to all of the targets. Instead, an attempt is made to construct an optimal fanout tree. SPGSNW calls SPLIT to choose a first target for the message. In addition to choosing this target, SPLIT divides the remaining targets between those to be processed by the first target and those to be processed by the root. The list of nodes for the target is appended to the message and sent. The rest of the nodes are iteratively processed by the root.

8. ERROR MESSAGES

Some utilities return error messages if called improperly. All such messages are syslogged, are in CAPITALS, and begin with the routine name. The messages are

GCAT: CALLED BY NON PARTICIPANT

GCAT: BUFFER LENGTH EXCEEDED

GOP: CALLED BY NON PARTICIPANT

GOP: LONG MESSAGE
Received too long a message

GOP: SHORT MESSAGE
Received too short a message

GRECVW: MESSAGE TRUNCATED

LRECVW: TRUNCATED MESSAGE

MESH: TOO LARGE
The mesh is too large to fit in the hypercube

SPGRCW: TRUNCATED MESSAGE

9. INTERNAL SUBROUTINES

The utilities library includes five subprograms that are used internally by the main procedures. They are documented here for completeness and because some of them may be useful in their own right. The modules are:

```
LOGICAL FUNCTION BETWEEN (I, J, K)
INTEGER I, J, K
```

BETWEEN decides whether node J is between node I and node K, that is whether node J lies on a shortest path in the hypercube topology from node I to node K. This function is used by SPLIT.

```
INTEGER FUNCTION GRAY (I)
INTEGER I
```

GRAY returns the Ith integer in the binary reflected gray code. (This code orders the node numbers so that consecutive numbers are adjacent nodes in a hypercube.)

```
INTEGER FUNCTION GINV (I)
INTEGER I
```

GINV is the inverse of GRAY. It returns the location of I in the gray code ordering.

```
SUBROUTINE SPLIT (NUMBER, LIST, FIRST)
INTEGER NUMBER, LIST(NUMBER), FIRST
```

SPLIT is used by SPGSNW (sparse global send). LIST is the list of NUMBER node numbers (**targets**) that must receive the message. Given the source node number r, the hypercube topology induces a partial ordering on the nodes where b<c is defined as b lies on a shortest path from r to c. SPLIT takes the list of targets, finds a minimal target node k, and then splits the remaining targets into the set of all nodes d for which k<d and the complement. FIRST is returned as the number of targets that are greater than k. These targets are put at the beginning of LIST, followed by initial target number k, followed by the

remaining targets.

```
      INTEGER FUNCTION XOR (I,J)
      INTEGER I, J
```

XOR returns the bitwise exclusive-or of integers I and J. Most FORTRAN compilers have a function that computes the exclusive-or but the name is different for different compilers. This function invokes the correct name. XOR is useful since XOR(I, 2**K) is the K-th neighbor of node I in the hypercube.

10. EXAMPLE PROBLEM

Suppose a vector X(N) is distributed among the P processors of a hypercube. For convenience assume that N = P*M. Assume that this vector must be normalized and then the entire vector must be available to node 0. The following code fragment will accomplish these tasks.

```
      INTEGER I, CI, GOTYPE, GSTYPE, GCTYPE, CUBEDIM, COPEN, MYNODE, CNT, ROOT
      DOUBLE PRECISION X(M), Y(N), DOT, NORM, DUMMY
      PARAMETER (DPSIZE = 8)
      DATA GOTYPE /100/, GSTYPE /200/, GCTYPE /300/
C
      CI = COPEN(MYPID())
C
C  Assume that X, M, and N have been
C     initialized
C
      ROOT = 0
C
C  Compute the local dot product
C
     DOT = 0
     DO 1 I = 1, M
          DOT = DOT + X(I)*X(I)
    1 CONTINUE
C
C  Sum local contributions in ROOT
C
     CALL GOP (CI, GOTYPE, DOT, 1, '+', ROOT, CUBEDIM(), DUMMY)
C
C  Distribute norm to all the processors
C
     IF (MYNODE() .EQ. ROOT) THEN
          NORM = DSQRT (DOT)
          CALL GSENDW (CI, GSTYPE, NORM, DPSIZE, CUBEDIM())
      ELSE
          CALL GRECVW (CI, GSTYPE, NORM, DPSIZE, CNT, CUBEDIM())
     ENDIF
C
C  Do the local normalization
C
     DO 2 I = 1, M
          X(I) = X(I)/NORM
    2 CONTINUE
C
C  Copy X to the concatenation buffer
C
```

```
      DO 3 I = 1,M
         Y(I) = X(I)
    3 CONTINUE
C
C  Concatenate the vector in ROOT
C
      CALL GCAT(CI, GCTYPE, Y, N*DPSIZE, M*DPSIZE, CNT, ROOT, CUBEDIM())
```

The array Y in Node 0 contains all of the vector X. Whether or not X is properly sorted depends on how the vector X was distributed on the nodes. If Node 0 contained X(I), I = 1, M and Node 1 contained X(I), I = M+1, 2*M etc., then X would be properly sorted.

Note that the naive way in which the norm is computed in this example can lead to problems with numerical underflow or overflow. Safer and more complicated implementations are possible and ought to be used in practice.

11. INDEX OF SUBROUTINES

BETWEEN	hypercube ordering
GCAT	global concatenation
GINV	gray code inverse
GOP	global commutative operation
GRAY	gray code
GRECVW	global receive
GSENDW	global send from node
LRECVW	long message receive
LSENDMSG	long message send from host
LSENDW	long message send from node
MESH	nearest neighbor mesh embedding
SPGRCW	sparse global receive
SPGSNW	sparse global send
SPLIT	node list separator
XOR	bitwise exclusive-or

12. REFERENCES

[1] J. E. Brandenburg and D. S. Scott
"Embeddings of Communication Trees and Grids into Hypercubes,"
iPSC Technical Report #1

13. FORTRAN LISTINGS

The following is a listing of the FORTRAN codes described in this report. There are three slightly different versions of the codes because three different FORTRAN compilers are of interest (f77 under Unix for use with the hypercube simulator, ftn286, and rmfort). The only distinctions between the versions are:

1. Whether integers are two bytes or four.
   f77 4 bytes
   ftn286 2 bytes
   rmfort 4 bytes

2. How byte addressable buffers are declared.
   f77 character*1
   ftn286 logical*1
   rmfort logical*1

3. How exclusive-or computations are implemented.
   see the listing for XOR

The actual listings given here are the rmfort version. All three versions are distributed with the operating system (as well as the C version), both as source and as libraries.

```
      LOGICAL FUNCTION BETWEEN(I, J, K)
      INTEGER I,J,K
c
c  BETWEEN decides whether node J lies on a shortest path from node I to
c  node K.  This function is used by SPGSNW and SPGRCW (sparse global send
c  and receive).
c
c  Calls:   XOR
c
      INTEGER XOR
      BETWEEN = XOR(I,K)  .EQ.  (XOR(I,J)  +  XOR(J,K))
      RETURN
      END
```

```
      SUBROUTINE GCAT (CI, TYPE, BUF, BUFLEN, MSGLEN, CNT, ROOT, DIM)
      INTEGER CI, TYPE, BUFLEN, MSGLEN, CNT, ROOT, DIM
c
c  byte indexed buffers are of type:
c
c        for ftn286 and RM FORTRAN:
      LOGICAL*1 BUF(BUFLEN)
c
c        for the simulator under f77:
c     CHARACTER*1 BUF(BUFLEN)
c
c  Global concatenation operation using spanning tree.  If the root
c    is zero then the messages are concatenated in numerical order.
c
c  All participating processes must have the same process id (PID).
c
c  Input..
c
c    CI          channel number (previously opened).
c    TYPE        type number.  must be the same for all participating
c                processes.  There must be no other messages of this type
c                OR OF TYPES TYPE+1, TYPE+2, ..., TYPE+DIM-1
c                in the system.
c    BUF         the input vector to be concatenated (which may actually
c                be any type).
c    BUFLEN      the length of BUF (in BYTES).
c    MSGLEN      the length of this contribution (in BYTES).
c    ROOT        Node id of root process (which will get the final message).
c                (if -32768, then the smallest node number in the active
c                subcube acts as root and then forwards the message to the host).
c    DIM         the size of the subcube participating.
c
c  Output..
c
c    BUF         for the root process, BUF contains the desired result.
c                for all other processes, BUF contains the partial result
c                for their subtrees.
c    CNT         the number of meaningful bytes in BUF.
c
c  Errors Conditions
c
c     If called by a nonparticipating node, an error message is put on
c       the syslog and then the subroutine exits.
c
c     If a result longer than BUFLEN BYTES is attempted an error
c       message is syslogged and the subroutine terminates.
c
c  Calls: MYNODE, MYPID, RECVW, SENDW, SYSLOG, XOR
c
      INTEGER BIT, CURLEN, DIFF, I, IGNORE, ME, MYNODE,
     *        MYPID, P, PARENT, PID, TROOT, XOR
c
      ME = MYNODE()
      P = 2**DIM
```

```
c
c     Find temporary root (either the real root, or the lowest
c     numbered node in the active subcube--found by zeroing the
c     DIM lowest bits in mynode).
c
      TROOT = MAX0((ME/P)*P, ROOT)
c
      CURLEN = MSGLEN
      PID = MYPID()
      DIFF = XOR(ME,TROOT)
      IF (DIFF .GE. P) THEN
         CALL SYSLOG(MYPID(),'GCAT: CALLED BY NON PARTICIPANT')
         RETURN
      ENDIF
c
c     Accumulate contributions from children, if any.
c
      BIT = 1

      DO 10 I = 1,DIM
         IF (MOD(DIFF,2*BIT) .NE. 0) GO TO 20
         CALL RECVW (CI,TYPE+I-1,BUF(CURLEN+1),BUFLEN-CURLEN,
     *        CNT,IGNORE,IGNORE)
c
         IF (CNT .GT. BUFLEN-CURLEN) THEN
            CALL SYSLOG(PID,'GCAT: BUFFER LENGTH EXCEEDED')
            RETURN
         ENDIF
c
         CURLEN = CURLEN + CNT
         BIT = 2*BIT
   10 CONTINUE
c
c  If exit normally, then this is the root.
c
      CNT = CURLEN
      IF (ROOT .LT. 0) CALL SENDW(CI, TYPE, BUF, CNT, -32768, PID)
      RETURN
c
c  Pass result back to parent.
c
   20 CONTINUE
      PARENT = XOR(ME, BIT)
      CALL SENDW(CI, TYPE+I-1, BUF, CURLEN, PARENT, PID)
      CNT = CURLEN
      RETURN
      END
```

```
      INTEGER FUNCTION GINV(N)
      INTEGER N
c
c     GINV = inverse of GRAY function
c
c     Answers the inevitable question, what task is being
c     carried out on node n?
c
      INTEGER T, XOR
      T = N
      GINV = N
    1 T = T/2
      IF (T .EQ. 0) RETURN
      GINV = XOR(GINV, T)
      GO TO 1
      END
```

```
      SUBROUTINE GOP (CI, TYPE, X, N, OP, ROOT, DIM, WORK)
      INTEGER CI, TYPE, N, ROOT, DIM
      CHARACTER*1 OP
      DOUBLE PRECISION X(N), WORK(N)
c
c  Global vector commutative operation using spanning tree.
c
c  All participating processes must have the same process id (PID).
c
c  Input..
c
c    CI          channel number (previously opened).
c    TYPE        message type.  Must be the same for all participating
c                processes.  There must be no other messages of this type
c                in the system.
c    X           the input vector to be used in the operation.
c    N           the length of the vector.
c    OP          '+'  sum
c                '*'  product
c                'M'  maximum
c                'm'  minimum
c    ROOT        Node id of root process (which will get the final message).
c                (if -32768, then the smallest node number in the active
c                subcube acts as root and then forwards the message to the host)
c    DIM         the size of the subcube participating.
c
c  Output..
c
c    X           for the root process, X contains the desired result.
c                for all other processes, X contains the partial result
c                for their subtrees.
c
c  Workspace
c
c    WORK        used to receive other contributions.
c
c  Errors Conditions
c
c         If called by a nonparticipating node, an error message is
c         syslogged and then the subroutine exits.
c
c         If a message longer than N elements is received, only the first N
c         elements are saved, an error message is syslogged,
c         and then the computation continues with the truncated results.
c
c         If a message shorter than N elements is received, then an error
c         message is syslogged and the computation continues.
c
c  Calls:  MYNODE, MYPID, RECVW, SENDW, SYSLOG, XOR
c
       INTEGER BIT, BYTES, CNT, DIFF, DPSIZE, I, IGNORE, ME, MYNODE,
     *   MYPID, P, PARENT, PID, TROOT, XOR
       PARAMETER (DPSIZE = 8)
```

```
c
      ME = MYNODE()
      P = 2**DIM
c
c     Find temporary root (either the real root, or the lowest
c     numbered node in the active subcube--found by zeroing the
c     DIM lowest bits in mynode).
c
      TROOT = MAX0((ME/P)*P, ROOT)
c
      PID = MYPID()
      DIFF = XOR(ME,TROOT)
      IF (DIFF .GE. P) THEN
         CALL SYSLOG(MYPID(),'GOP: CALLED BY NON PARTICIPANT')
         RETURN
      ENDIF
c
c     Accumulate contributions from children, if any
c
      BIT = P/2
      BYTES = DPSIZE*N
    5 IF (BIT .LE. DIFF) GO TO 20
         CALL RECVW(CI,TYPE,WORK,BYTES,CNT,IGNORE,PID)
         IF (CNT .GT. BYTES) CALL SYSLOG(PID,'GOP: LONG MESSAGE')
         IF (CNT .LT. BYTES) CALL SYSLOG(PID,'GOP: SHORT MESSAGE')
         DO 10 I = 1, N
            IF (OP .EQ. '+') X(I) = X(I) + WORK(I)
            IF (OP .EQ. '*') X(I) = X(I) * WORK(I)
            IF (OP .EQ. 'M') X(I) = DMAX1(X(I),WORK(I))
            IF (OP .EQ. 'm') X(I) = DMIN1(X(I),WORK(I))
   10    CONTINUE
         BIT = BIT/2
      GO TO 5
c
c     Pass result back to parent
c
   20 CONTINUE
      IF (BIT .NE. 0) THEN
         PARENT = XOR(ME, BIT)
         CALL SENDW(CI,TYPE,X,BYTES,PARENT,PID)
      ELSE
         IF (ROOT .LT. 0) CALL SENDW(CI,TYPE,X,BYTES,-32768,PID)
      ENDIF
      RETURN
      END
```

```
      INTEGER FUNCTION GRAY(J)
      INTEGER J, XOR
C
C     GRAY(J) = J-th entry in gray code
C     domain and range both 0..127
C
C     This is the standard way to obtain a one-dimensional mesh
C     or ring of tasks on the hypercube.  If the j-th task is
C     carried out on node number GRAY(J), then each task is
C     connected directly to its predecessor and its successor
C     in the hypercube topology.
C
      GRAY = XOR(J, J/2)
      RETURN
      END
```

```
      SUBROUTINE GRECVW(CI, TYPE, BUF, LEN, CNT, DIM)
      INTEGER CI, TYPE, BUF(*), LEN, CNT, DIM
c
c  Global send participant.  Receives message from unknown source and
c       sends it on to some neighbors.
c
c  All participating processes must have the same process id (PID).
c
c  Input..
c
c    CI          channel number (previously opened).
c    TYPE        message type.  Must be the same for all participating
c                processes.  There must be no other messages of this type
c                in the system.
c    LEN         the length of BUF in BYTES.
c    DIM         the dimension of the subcube participating in the send.
c
c  Output..
c
c    BUF         the message (which may actually be any type).
c    CNT         the length (in BYTES) of the message received.
c
c  Error Conditions
c
c    If a message longer than LEN bytes is received then only
c    LEN bytes will be stored in BUF and the rest of the
c    message will be lost.  In this case an error message
c    will be sent to syslog but the remnants of the message
c    will be sent on.
c
c    NOTE: only those nodes which will participate in the
c        send can call GRECVW.  Any other node which calls it
c        will never return.
c
c  Calls: MYNODE, MYPID, RECVW, SENDW, SYSLOG, XOR
c
      INTEGER BIT, I, LENOUT, ME, MYNODE, MYPID, NODE, P, PID,
     *   PIDIN, XOR
c
      P = 2**DIM
      ME = MYNODE()
      PID = MYPID()
      CALL RECVW(CI, TYPE, BUF, LEN, CNT, NODE, PIDIN)
c
      LENOUT = CNT
      IF (CNT .GT. LEN) THEN
         CALL SYSLOG(PID,'GRECVW: MESSAGE TRUNCATED')
         LENOUT = LEN
      ENDIF
c
      BIT = 2*XOR(ME,NODE)
c
c  Check to see if received from host.
c
```

```
      IF (NODE .LT. 0) BIT = 1
C
      DO 10 I = 1, DIM
         IF (BIT .EQ. P) RETURN
         NODE = XOR(ME,BIT)
         CALL SENDW(CI, TYPE, BUF, LENOUT, NODE, PID)
         BIT = 2*BIT
   10 CONTINUE
      END
```

```
      SUBROUTINE GSENDW(CI, TYPE, BUF, LEN, DIM)
      INTEGER CI, TYPE, BUF(*), LEN, DIM
c
c  Global send of data.  Other participants call grecvw.
c
c  All participating processes must have the same process id (PID).
c
c  Input
c
c    CI          channel number (previously opened).
c    TYPE        message type.  Must be the same for all participating
c                processes.  There must be no other messages of this type
c                in the system.
c    BUF         the message buffer (which may actually be any type).
c    LEN         the length of the buffer in BYTES
c    DIM         the dimension of the subcube
c
c  Calls:   MYNODE, MYPID, SENDW, XOR
c
      INTEGER BIT, I, ME, NODE, MYNODE, MYPID, PID, XOR
c
      ME = MYNODE()
      PID = MYPID()
      BIT = 1
c
      DO 10 I = 1, DIM
         NODE = XOR(ME,BIT)
         CALL SENDW(CI, TYPE, BUF, LEN, NODE, PID)
         BIT = 2*BIT
   10 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LRECVW(CI, TYPE, BUF, LEN4, CNT4, NODE, PID)
      INTEGER CI, TYPE, NODE, PID
      INTEGER*4 LEN4, CNT4
c
c  byte indexed buffers are of type:
c
c        for ftn286 and RM FORTRAN:
      LOGICAL*1 BUF(*)
c
c        for the simulator under f77:
c     CHARACTER*1 BUF(*)
c
c send/recv cannot process messages larger than 16K bytes.  LSENDW and
c LRECVW can be used to send large messages between nodes.  LSENDMSG and
c LRECVW can be used to send large messages from the host to a node.
c There is no subroutine to receive large messages on the host since
c the semantics of type is different for RECVMSG.  The message
c is sent in 16K byte chunks and terminated with a smaller chunk.  If
c the message is an even multiple of 16K bytes, this requires sending
c a zero byte message as a terminator and requires that LEN4 >= CNT4 + 1
c
c  Input..
c
c    CI         channel number (previously opened).
c    TYPE       message type.  There must be no other messages of this type
c               received by the target node.
c    BUF        the message buffer (which may actually be any type).
c    LEN4       an INTEGER*4, the length of BUF in BYTES.
c
c  Output..
c
c    CNT4       an INTEGER*4, the length of the received message in BYTES.
c    NODE       the source node.
c    PID        the source pid.
c
c  Calls: RECVW, SYSLOG
c

      INTEGER*4 REMAIN, INDEX
      INTEGER LEN2, MYPID, CNT2
      PARAMETER (MAXLEN = 16384)
c
      REMAIN = LEN4
      INDEX = 1
      CNT4 = 0
      LEN2 = MAXLEN
c
```

```
   10 CONTINUE
        IF (LEN4 - INDEX + 1 .LT. MAXLEN) LEN2 = LEN4 - INDEX + 1
        CALL RECVW(CI, TYPE, BUF(INDEX), MAXLEN, CNT2, NODE, PID)
        IF (CNT2 .GT. LEN2)
      *    CALL SYSLOG(MYPID(),'LRECVW: TRUNCATED MESSAGE')
        INDEX = INDEX + CNT2
        CNT4 = CNT4 + CNT2
      IF (CNT2 .EQ. MAXLEN) GO TO 10
C
      RETURN
      END
```

```
      SUBROUTINE LSENDMSG(CI, TYPE, BUF, LEN4, NODE, PID)
      INTEGER CI, TYPE, NODE, PID
      INTEGER*4 LEN4
c
c  byte indexed buffers are of type:
c
c        for ftn286 and RM FORTRAN:
      LOGICAL*1 BUF(*)
c
c        for the simulator under f77:
c     CHARACTER*1 BUF(*)
c
c see LRECVW for details
c
c  Input..
c
c    CI         a channel number (previously opened).
c    TYPE       message type.  Must be the same as the corresponding
c               LRECVW call.
c    BUF        the message buffer (which may actually be any type).
c    LEN4       an INTEGER*4 which is the length of the message in BYTES.
c    NODE       the destination node.
c    PID        the destination pid.
c
c  Calls SENDMSG
c
      INTEGER*4 REMAIN, INDEX
      INTEGER I, LEN2, MAXLEN, NUMMSG
      PARAMETER (MAXLEN = 16384)
c
      REMAIN = LEN4
      INDEX = 1
      LEN2 = MAXLEN
      NUMMSG = LEN4/MAXLEN + 1
c
      DO 10 I = 1, NUMMSG
c
c special settings for last message
c
         IF(REMAIN .LT. MAXLEN) LEN2 = REMAIN
         IF(REMAIN .EQ. 0) INDEX = 1
         CALL SENDMSG(CI, TYPE, BUF(INDEX), LEN2, NODE, PID)
         REMAIN = REMAIN - LEN2
         INDEX = INDEX + LEN2
   10 CONTINUE
c
      RETURN
      END
```

```
      SUBROUTINE LSENDW(CI, TYPE, BUF, LEN4, NODE, PID)
      INTEGER CI, TYPE, NODE, PID
      INTEGER*4 LEN4
c
c  byte indexed buffers are of type:
c
c        logical*1 for ftn286 and RM FORTRAN
      LOGICAL*1 BUF(*)
c
c        character*1 for f77 and the simulator
c     CHARACTER*1 BUF(*)
c
c  see LRECVW for details
c
c  Input..
c
c    CI          a channel number (previously opened).
c    TYPE        message type.  Must be the same as the corresponding
c                LRECVW call.
c    BUF         the message buffer (which may actually be any type).
c    LEN4        an INTEGER*4 which is the length of the message in BYTES.
c    NODE        the destination node.
c    PID         the destination pid.
c
c  Calls:  sendw
c
      INTEGER*4 REMAIN, INDEX
      INTEGER I, LEN2, NUMMSG
      PARAMETER (MAXLEN = 16384)
c
      REMAIN = LEN4
      INDEX = 1
      LEN2 = MAXLEN
      NUMMSG = LEN4/MAXLEN + 1
c
      DO 10 I = 1, NUMMSG
c
c special parameters for last message
c
         IF(REMAIN .LT. MAXLEN) LEN2 = REMAIN
         IF(REMAIN .EQ. 0) INDEX = 1
c
         CALL SENDW(CI, TYPE, BUF(INDEX), LEN2, NODE, PID)
         REMAIN = REMAIN - LEN2
         INDEX = INDEX + LEN2
   10 CONTINUE
c
      RETURN
      END
```

```
      SUBROUTINE MESH(ME, DIM, LINDIM, PERBC, COORDS, PRED, SUCC,
     *    WORK)
      INTEGER ME, DIM, LINDIM(DIM), PERBC(DIM), COORDS(DIM),
     *    PRED(DIM), SUCC(DIM), WORK(DIM)
c
c  MESH     Embeds a rectangular mesh into the hypercube.
c           The location of the current node in the mesh is computed
c           as well as the neighboring node numbers.
c           If mesh is called with too large a mesh, some of the
c           neighboring numbers may not exist in the cube.
c
c Input..
c
c    ME             the node number of interest.
c    DIM            the physical dimension of the mesh (usually 1, 2, or 3).
c    LINDIM         the linear dimension (number of processors) in each
c                   direction of the mesh.
c    PERBC          a flag indicating whether a particular direction has
c                   periodic boundary conditions.  perbc should be set as
c                   follows:
c
c                   PERBC(I) = 0     for non periodic (terminated) boundaries
c                   PERBC(I) = 1     for periodic boundaries
c
c Output..
c
c    COORDS         Gives the coordinates of me in the mesh,
c                   starting with 0 so that the largest the ith coordinate
c                   can be is LINDIM(I)-1.  If me is not part
c                   of the mesh all coordinates are returned as -1.
c    PRED           PRED(I) is the node number proceding me in the ith
c                   direction.  -1 is used when there is no predecessor.
c    SUCC           SUCC(I) is the node number succeeding me in the ith
c                   direction.  -1 is used when there is no successor.
c
c Workspace
c
c    WORK           holds the result of rounding the elements of LINDIM
c                   up to the next power of two.
c
c  Calls:  CUBEDIM, GRAY, GINV, MYPID, SYSLOG
c
      INTEGER CUBEDIM, D, GRAY, GINV, I, J, MYPID, SIZE, SLICE, TEMP
c
c  Round up LINDIM to next power of 2.
c
      SIZE = 1
      DO 1 I = 1, DIM
         D = 0
         IF(LINDIM(I) .GT. 1) D = ALOG(FLOAT(LINDIM(I)-1))/ALOG(2.0)+1
         WORK(I) = 2**D
         SIZE = SIZE*WORK(I)
    1 CONTINUE
```

```
c
c  Check size of hypercube.
c
      IF(SIZE .GT. 2**CUBEDIM())
     *  CALL SYSLOG(MYPID(),'MESH: TOO LARGE')
c
c  Decompose ME.
c
      TEMP = ME
      DO 2 I = 1, DIM
         COORDS(I) = MOD(TEMP,WORK(I))
         COORDS(I) = GINV(COORDS(I))
         TEMP = TEMP/WORK(I)
    2 CONTINUE
c
c  Kill nodes outside mesh.
c
      DO 4 I = 1,DIM
         IF(TEMP .GT. 0 .OR. COORDS(I) .GE. LINDIM(I)) THEN
            DO 3 J = 1,DIM
               COORDS(J) = -1
               PRED(J) = -1
               SUCC(J) = -1
3           CONTINUE
            RETURN
         ENDIF
4     CONTINUE
c
c  Set PRED and SUCC.
c
      SLICE = 1
      DO 5 I = 1, DIM
         TEMP = GRAY(COORDS(I)-1) - GRAY(COORDS(I))
         PRED(I) = ME + TEMP*SLICE
         TEMP = GRAY(COORDS(I)) - GRAY(COORDS(I)+1)
         SUCC(I) = ME - TEMP*SLICE
         SLICE = SLICE*WORK(I)
    5 CONTINUE
c
c  Fix boundary conditions.
c
      SLICE = 1
      DO 6 I = 1, DIM
         IF(PERBC(I) .EQ. 0) THEN
            IF(COORDS(I) .EQ. 0) PRED(I) = -1
            IF(COORDS(I) .EQ. LINDIM(I)-1) SUCC(I) = -1
          ELSE
            IF(COORDS(I) .EQ. 0) PRED(I) = ME + GRAY(LINDIM(I)-1)*SLICE
            IF(COORDS(I) .EQ. LINDIM(I)-1)
     *         SUCC(I) = ME - GRAY(LINDIM(I)-1)*SLICE
         ENDIF
         SLICE = SLICE*WORK(I)
    6 CONTINUE
c
      RETURN
      END
```

```
      SUBROUTINE SPGRCW(CI, TYPE, BUF, LEN, CNT)
      INTEGER CI, TYPE, BUF(*), LEN, CNT
c
c   This subroutine participates in a sparse global send.  It receives
c   a message with a list of node numbers appended which must also
c   receive the message.  It calls SPGSNW to forward the message.
c   The process is started by a call to SPGSNW.
c
c  All participating processes must have the same process id (PID).
c
c  Input..
c
c    CI         channel number (previously opened).
c    TYPE       message type.  Must be the same for all participating
c               processes.  There must be no other messages of this type
c               in the system.
c    LEN        the length of BUF in BYTES.  LEN must be larger than the
c               expected message.  In particular, the buffer must be able
c               to hold 2**CUBEDIM() additional integers.
c
c  Output..
c
c    BUF        contains the message.
c    CNT        the length of the received message in BYTES.
c
c  Error Conditions
c
c    If the received message is truncated then an error message is
c    syslogged and the subroutine returns.
c
c  Calls:  MYPID, RECVW, SPGSNW, SYSLOG
c
      INTEGER MYPID, NODE, NUMBER, PID, PIDIN
      PARAMETER (INTSIZE = 4)
c
      PID = MYPID()
      CALL RECVW(CI, TYPE, BUF, LEN, CNT, NODE, PIDIN)
      IF (CNT .GT. LEN) THEN
         CALL SYSLOG(PID, 'TRUNCATED MESSAGE IN SPGRCW')
         RETURN
      ENDIF
c
      NUMBER = BUF(CNT/INTSIZE)
      CNT = CNT - INTSIZE*(NUMBER + 1)
      CALL SPGSNW(CI, TYPE, BUF, CNT, BUF(CNT/INTSIZE + 1), NUMBER)
      RETURN
c
      END
```

```
      SUBROUTINE SPGSNW(CI, TYPE, BUF, LEN, LIST, NUMBER)
      INTEGER CI, TYPE, BUF(*), LEN, LIST(*), NUMBER
c
c   This subroutine sends a buffer to a subset of nodes specified by
c   an index list.  All the nodes which are to receive the buffer must
c   call SPGRCW.
c
c   All participating processes must have the same process id (PID).
c
c   Input..
c
c     CI          channel number (previously opened).
c     TYPE        message type.  Must be the same for all participating
c                 processes.  There must be no other messages of this type
c                 in the system.
c     BUF         the message buffer (which may actually be any type).
c     LEN         the number of BYTES of data in BUF.  SPGRCW can only
c                 receive messages that are a multiple of INTSIZE.
c                 Therefore LEN must be a multiple of INTSIZE.
c                 Furthermore the LIST is appended to the message and so
c                 BUF must have length at least LEN + INTSIZE*NUMBER.
c     LIST        the list of node numbers which should receive the data.
c     NUMBER      the number of node numbers in LIST.
c
c   Output..
c
c      the list of node numbers is destroyed
c
c  Calls:  MYPID, SENDW, SPLIT
c
      PARAMETER (INTSIZE = 4)
      INTEGER BUFLEN, FIRST, I, J, MYPID, NODE, PID
c
      PID = MYPID()
      BUFLEN = LEN/INTSIZE
   10 IF (NUMBER .EQ. 0) RETURN
         CALL SPLIT(NUMBER, LIST, FIRST)
         NODE = LIST(FIRST+1)
         DO 20 J = 1, FIRST
     BUF(BUFLEN + J) = LIST(J)
   20    CONTINUE
c
         BUF(BUFLEN + FIRST + 1) = FIRST
         CALL SENDW(CI, TYPE, BUF, LEN + INTSIZE*(FIRST+1), NODE, PID)
         J = 1
         DO 30 I = FIRST+2, NUMBER
           LIST(J) = LIST(I)
           J = J+1
   30    CONTINUE
c
         NUMBER = NUMBER - FIRST - 1
      GO TO 10
c
      END
```

```
      SUBROUTINE SPLIT(NUMBER, LIST, FIRST)
      INTEGER NUMBER, LIST(*), FIRST
c
c SPLIT takes a set of node numbers, finds a child node (one with
c   no other node between me and it), and then separates the remaining
c   nodes into the descendents of the child (all nodes for which the
c   child is between me and the node) and the rest.  The descendents
c   are put first, the child in the middle and the rest afterward.
c
c  Input..
c
c    NUMBER     the number of node numbers.
c    LIST       the list of node numbers.
c
c  Output..
c
c    LIST       sorted so that the decendents of the child are first,
c               the child in the middle, and the rest last.
c    FIRST      returned as the number of decendents of the child.
c
c  Calls: MYNODE
c
      INTEGER CHILD, HI, KEY, LO, ME, MYNODE, TEMP, TEST
      LOGICAL BETWEEN
      ME = MYNODE()
c
c find key
c
      KEY = 1
      TEST = 2
   10 IF (TEST .GT. NUMBER) GO TO 20
 IF (BETWEEN(ME, LIST(TEST), LIST(KEY))) KEY = TEST
 TEST = TEST+1
 GO TO 10
c
c swap key to front
c
   20 CHILD = LIST(KEY)
      LIST(KEY) = LIST(1)
c
c split rest of list.  put descendents of key first.
c
      LO = 2
      HI = NUMBER
c
c move lo upward
c
   30 IF (LO .GT. HI) GO TO 60
        IF (.NOT. BETWEEN(ME, CHILD, LIST(LO))) GO TO 40
        LO = LO + 1
      GO TO 30
c
c move hi downwards
c
```

```
   40 IF (LO .GE. HI) GO TO 60
        IF (BETWEEN(ME, CHILD, LIST(HI))) GO TO 50
        HI = HI - 1
      GO TO 40
c
c swap
c
   50 TEMP = LIST(LO)
      LIST(LO) = LIST(HI)
      LIST(HI) = TEMP
      LO = LO + 1
      HI = HI - 1
      GO TO 30
c
c put child in the middle
c
   60 FIRST = LO - 2
      LIST(1) = LIST(FIRST+1)
      LIST(FIRST+1) = CHILD
      RETURN
      END
```

```
      INTEGER FUNCTION XOR(M,N)
c
c  exclusive or
c
c  Pick one of the following:
c
c  UNIX 4.2, f77:
c     XOR = OR(M,N)-AND(M,N)
c
c  Intel FTN286:
c     XOR = M.NEQV.N
c
c  Ryan-McFarland Fortran
      XOR = IEOR(M,N)
c
      RETURN
      END
```

Printed in USA/SC6054/0886/1k/WCP/VJC
Order Number 280323-001

INTEL SCIENTIFIC COMPUTERS
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
(503) 629-7629